Gestor de almacenamiento en redes P2P

Olías de Lima Pancorbo, Carlos
García Sánchez, José Daniel (Tutor)

AGRADECIMIENTOS

Quiero expresar mi agradecimiento a todas las personas que han hecho posible este enorme proyecto, desde tutor Daniel García Sánchez, que ha seguido su desarrollo desde el principio, el “Club .Net”, que ayudó con mi iniciación en esta plataforma de desarrollo, gente anónima de los foros de Msdn, resolviendo incansablemente mis dudas y a los profesores que han aportado todos los conocimientos en los que el proyecto se basa.

Así mismo me gustaría agradecer a los compañeros de clase el tiempo que hemos pasado juntos en un pequeño pero unido grupo.

Por último no quisiera olvidarme de familiares y amigos, que me han apoyado en el tiempo que ha durado la realización del proyecto.

ÍNDICE

1. INTRODUCCIÓN

1.1. MARCO GENERAL

Los requerimientos de las grandes empresas y de los usuarios son cada vez mayores, y la tecnología, para satisfacer esta demanda, avanza a pasos agigantados día a día. Prueba de ello es la Ley de Moore (ley empírica), que expresa que "*aproximadamente cada dos años se duplica el número de transistores en un computador*", cuyo cumplimiento se ha podido constatar hasta hoy, lo que es una clara muestra de este enorme desarrollo.

Pero aún con más intensidad que la potencia de los procesadores está avanzando la capacidad de las redes de comunicación. Tanto es así, que si antes se consideraba como principal "cuello de botella" las redes de comunicación en el desarrollo de aplicaciones distribuidas o aplicaciones con comunicación a través de la red, ahora el problema empieza a ser los propios elementos físicos de la informática, el *hardware*.

George Gilder, director del Media Lab en Massachussets y reconocido tecnólogo, ya dejó entrever dicho crecimiento de la capacidad en las comunicaciones en su libro "Life after television". Las aseveraciones de Gilder le llevaron a formular lo que se conoce como "Ley de Gilder" o "Ley de la Banda Ancha" que dice que "*la capacidad de las comunicaciones se triplica cada doce meses*", y que es una muestra más del potencial de crecimiento de las TIC. Un claro ejemplo del planteamiento de Gilder que demuestra dicho potencial es que ya en 2001 era posible enviar más información por un solo cable de comunicaciones que toda la información que se enviaba por Internet en 1997. Y este crecimiento sigue de forma continuada, rompiéndose casi a diario records de velocidad en transmisión de datos, siendo uno de los últimos el logrado por "Alcatel" en los cables de transmisión submarina que conectan todo el mundo, llegándose a alcanzar en 2009 una velocidad de 100 *petabits* por segundo.

Y si bien la necesidad de información de los usuarios de las redes de comunicación crecen año a año, uno de los elementos determinantes para este espectacular crecimiento de las redes y su capacidad ha sido la popularización del

software P2P, y la demanda de sus usuarios de capacidades de transmisión cada vez mayores.

Muestra de este increíble crecimiento es el crecimiento en España de las redes de comunicaciones. Cuando se empezó a popularizar el P2P gracias al famoso *Napster*, las conexiones a nivel usuario en España eran del orden de los 28 *kbps*. Estas conexiones eran caras, muy inestables, y con una baja penetración a nivel doméstico. En 2007, siete años después, se ofrecían conexiones de 20 *Mbps*, del orden mil veces más veloces, a pesar de las posiciones monopolísticas que han tenido algunas empresas suministradoras de *Internet*, que han dificultado una mayor competencia y han bloqueado desde hace años la capacidad de subida de datos en las líneas ADSL domésticas, impidiendo así un mayor crecimiento del que disponen otros países como por ejemplo Japón, con conexiones diez veces superiores a las nuestras, o incluso nuestro vecino Portugal, que en 2009 ofrece conexiones de 1Gbps simétricos.

A pesar de ser una tecnología relativamente incipiente, el uso del P2P para compartir (de forma legal en España) música, vídeo u otros archivos en formato digital ha popularizado tanto su uso que se estima más de la mitad de ancho de banda consumido en el planeta lo consumen programas que hacen uso de dicha tecnología. Pero no solo se comparten archivos con contenidos audiovisuales gracias a esta tecnología, su desarrollo ha derivado en otros interesantes conceptos basados en P2P, como televisión a través de P2P, telefonía *VOIP* (el conocido Skype), compartir "ciclos de procesador" en aplicaciones distribuidas (SETI@Home, por ejemplo)…

Este proyecto pretende seguir explorando las posibilidades que ofrece el crecimiento de las redes de comunicaciones y la tecnología P2P, explotando vías relativamente nuevas como son los sistemas de archivos distribuidos mediante P2P.

1.2. OBJETIVO DEL PROYECTO

El proyecto va a consistir en la implementación de un sistema de archivos distribuido mediante P2P, basándose en el entorno de desarrollo *.NET*.

Para dicho desarrollo, la primera parte del proyecto consiste en la familiarización con un nuevo lenguaje de desarrollo integrado en la plataforma *Visual Studio: Visual Basic.NET*. Se ha optado por este lenguaje ya que existe gran documentación sobre él, y es capaz de acceder a todas las librerías que proporciona .NET. Este entorno de desarrollo tiene una "orientación a objetos" de la que hasta este momento solo se había obtenido conocimientos teóricos en una o dos asignaturas de la carrera cursada (Ingeniería Técnica en Informática de Gestión), lo cual supone un considerable esfuerzo de aprendizaje, así como un paso más en la aplicación de los conocimientos adquiridos en la misma. Dicho esfuerzo culmina con un pequeño manual de este lenguaje de programación (Anexo A)

En segundo lugar, una vez adquiridos los conocimientos necesarios para el desarrollo de aplicaciones en dicha plataforma, se implementará un Sistema de Archivos Distribuidos mediante tecnología P2P. Este software permitirá la replicación de los archivos seleccionados por un usuario en nodos (ordenadores) ajenos, permitiendo recuperarlos o eliminarlos según se requiera. Cualquier terminal conectado como nodo a la red de archivos distribuida, independientemente de su ubicación, es capaz de realizar estas tres funciones: replicar archivos en nodos ajenos, eliminarlos, o recuperarlos.

La idea básica es una vuelta de tuerca al concepto clásico del software P2P: si originalmente los P2P han sido usados para compartir archivos personales con otros usuarios de la red, en este caso, y apoyándonos en tecnologías P2P, lo que compartimos es espacio de almacenamiento. Y gracias a que otros usuarios también comparten una porción de su espacio de almacenamiento, podemos realizar replicas de nuestros archivos personales en él, teniendo acceso a nuestros archivos personales cuando y donde queramos, solo limitados por la capacidad de la red mediante la que nos conectemos.

Así pues, un usuario que desee utilizar esta aplicación podrá crear un nuevo usuario, conectarse con él a la red y compartir un cierto espacio de su disco duro. A partir de ese momento, otros usuarios podrán replicar sus archivos personales en su

“espacio compartido”, al mismo tiempo que él podrá replicar sus propios archivos en usuarios conectados a la red, recuperar los archivos si están conectados los usuarios replicantes de los mismos, borrarlos, etc. Dichos archivos, independientemente del tamaño, son fraccionados y cifrados, para que solo el usuario legítimo del archivo, y mediante la “llave” correcta pueda ser capaz recuperarlos, añadiendo un elemento extra de seguridad al protocolo. Dicho cifrado sirve, además, para que el servidor central que maneja la red no tenga conocimiento de los archivos que por la red circulan.

En definitiva, lo que se pretende con este proyecto es desarrollar un nuevo concepto de software P2P, tocando todos los temas fundamentales de estas redes (fragmentación, distribución de tareas, cifrado, protocolos de conexión, compartir recursos, etc), y aplicando los conocimientos teóricos adquiridos en la Ingeniería Técnica en Informática de Gestión y muchos otros adquiridos durante la realización de este proyecto, demostrando que este nuevo concepto en el universo de los programas P2P es factible.

1.3. DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS

A continuación se ofrece un listado de los acrónicos y abreviaturas que aparecen en este documento:

- .NET: Referido a la plataforma de desarrollo Visual Studio .Net, o a los lenguajes que este entorno de desarrollo proporciona.
- A.I.C.H.: Gestión Avanzada Inteligente de Corrupción. Mecanismo usado por el eMule para evitar la corrupción de las descargas, mejora del I.C.H.
- ARPANET: Advanced Research Projects Agency Network. Red de computarores creada por encargo del Departamento de Defensa de los Estados Unidos como medio de comunicación para los diferentes organismos del país.
- BCL: Bibliotecas de clases de .Net
- CLI: Infraestructura de lenguaje común (*Common Language Infrastructure*). Su principal característica es la de permitir que aplicaciones escritas en distintos lenguajes de alto nivel puedan luego ejecutarse en múltiples plataformas tanto de hardware como de software sin necesidad de reescribir o recompilar su código fuente
- CLR: Entorno Común de Ejecución para Lenguajes
- CPU: Acrónimo inglés de "Unidad central de procesamiento" o simplemente el procesador o microprocesador.
- CTS: Sistema Genérico de Tipos, del framework de .NET.
- DHT: Tablas de Hash Distribuido.
- Firewall: Cortafuegos (*firewall,* en inglés). Es un elemento de hardware o software utilizado en una red de computadoras para controlar las comunicaciones, permitiéndolas o prohibiéndolas según las políticas de red
- Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado. En referencia al documento, framework se refiere al framework de .Net.
- GPL: La Licencia Pública General. Licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software.
- Hardware: Parte física del ordenador.
- Hash: Se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc.

- Hub: Un concentrador o hub es un dispositivo que permite centralizar el cableado de una red y poder ampliarla. Esto significa que dicho dispositivo recibe una señal y repite esta señal emitiéndola por sus diferentes puertos.
- Internet: Es un método de interconexión descentralizada de redes de computadoras implementado en un conjunto de protocolos denominado TCP/IP y garantiza que redes físicas heterogéneas funcionen como una red lógica única, de alcance mundial.
- I.C.H.: Gestión Inteligente de Corrupción. Mecanismo usado por el eMule para evitar la corrupción de las descargas.
- IP: Una dirección IP es un número que identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (*Internet Protocol*).
- ISP: Proveedor de servicios de Internet.
- JIT o Just-in-time: Compilación en tiempo de ejecución.
- Kbps: Kilo bit por segundo.
- Leechers: En el protocolo BitTorrent, se denomina así a todos los usuarios que están en la red descargando el archivo pero que todavía no tienen el archivo completo.
- Llave: Referído a la clave que se usa para cifrar o descifrar un "mensaje".
- Mainframe: Una computadora central, grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos.
- Mbps: Mega bit por segundo.
- MIMD: Multiple Instruction, Multiple Data streams. Referente a la "Taxonomía de Flynn", varios procesadores autónomos que ejecutan simultáneamente instrucciones diferentes sobre datos diferentes.
- MISD: Multiple Instruction, Single Data streams. Referente a la "Taxonomía de Flynn", arquitectura poco común, usada para la tolerancia a fallos.
- Mp3: MPEG-1 Audio Layer 3, más conocido como MP3. Es un formato de audio digital comprimido con pérdida.
- MSIL: Acrónimo de Microsoft Intermediate Language. Es un *bytecode* (código intermedio más abstracto que el código máquina) que la Tecnología .NET de Microsoft utiliza para lograr independencia de la plataforma y seguridad en ejecución.
- Nodo: Referido a una red de ordenadores, es cada terminal que la compone.
- P2P: *Peer-to-peer*.

- Peer-to-peer: Una red punto a punto, que no tiene ni clientes ni servidores fijos.
- Peers: En el protocolo BitTorrent, se denomina así a todos los usuarios que están en la red.
- Petabits: Unidad de medida. Mil billones de bits.
- Proxy: Un programa o dispositivo que realiza una acción en representación de otro. La finalidad más habitual es la de servidor proxy, que sirve para permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, esto es, una única dirección IP.
- Relé: Es un dispositivo electromecánico, que funciona como un interruptor controlado por un circuito eléctrico.
- RIAA: Recording Industry Association of America, asociación estadounidense que representa a la mayor parte de la industria discográfica.
- Seeds: En el protocolo BitTorrent, son los usuarios de la red que poseen el archivo completo.
- SIMD: Single Instruction, Multiple Data streams. Referente a la "Taxonomía de Flynn", un computador que explota varios flujos de datos dentro de un único flujo de instrucciones para realizar operaciones que pueden ser paralelizadas de manera natural.
- SISD: Single Instruction, Single Data stream. Referente a la "Taxonomía de Flynn", computador secuencial que no explota el paralelismo en las instrucciones ni en flujos de datos.
- Software: Es la parte intangible de una computadora, es decir, los programas.
- TCP: *Transmission Control Protocol*. Es uno de los protocolos fundamentales de Internet. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.
- TIC: Tecnologías de la información y la comunicación.
- Tracker: En el protocolo BitTorrent, es un servidor especial que contiene la información necesaria para que los peers se conecten unos con otros.
- TVP2P: Tecnología que apoyada en protocolos P2P, permite ver la televisión por Internet en directo.
- VOIP: También llamado voz sobre IP, es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP.

2. ESTADO DE LA CUESTIÓN

2.1. REDES Y SISTEMAS DISTRIBUIDOS

2.1.1. INTRODUCCIÓN

Los sistemas distribuidos y las redes de comunicaciones que se utilizan en la transmisión de datos están íntimamente ligados en todo su proceso evolutivo. Para entender éste, nos tenemos que remontar a sus orígenes, cuando la computación con fines prácticos daba sus primeros pasos, entre 1940 y 1952, cuando los ordenadores ocupaban plantas enteras de un edificio, estaban compuestos por válvulas y había que programarlos modificando a mano los valores directamente en los "circuitos". Aunque Jorge Stibz construyó en 1949 en los laboratorios Bell una máquina programable que trabajaba con números complejos, el Complex Calculator, se considera que el primer ordenador fue desarrollado en 1941, el Z3 del alemán Konrad Zuse. Le siguió en 1944 el Mark I de Howard Aiken y Grace Hopper, construido en la Universidad de Harvard con la colaboración de IBM. Pesaba cinco toneladas, tenía más de 750000 piezas y 800 km de cable. La sustitución de los relés por tubos de vacío dio lugar a la Primera Generación de ordenadores electrónicos, y aunque eran de uso estrictamente científico y militar fue el primer paso de la revolución que hoy en día conocemos. El primer computador fue fabricado en 1945, el ENIAC (Electronic Numerical Integrator and Calculator) de los estadounidenses John Eckert y John Mauchly que se usó en el cálculo de trayectorias de proyectiles.

Se inventa el transistor, y aparecen los primeros ordenadores comerciales, es la llamada "Segunda Generación". Se comienza a reducir el tamaño y consumo, al mismo tiempo que se aumenta de forma drástica la capacidad de proceso. En 1964 la aparición del IBM 360 marca el comienzo de la tercera generación. Las placas de circuito impreso con múltiples componentes pasan a ser reemplazadas por los circuitos integrados.

La evolución de la computación avanza a pasos acelerados y, junto con esta revolución, surge la necesidad de conectar ordenadores para la transmisión de datos. Aunque la idea de una red de computadoras diseñada para permitir la comunicación general entre usuarios de varias computadoras ya existía años atrás, en octubre de

1962, J.C.R. Licklider, pionero en lo que se refiere a una red mundial, fue nombrado jefe de la oficina de procesado de información DARPA.

Mientras tanto, Paul Baran estaba trabajando desde 1959 en la RAND Corporation en una red segura de comunicaciones capaz de sobrevivir a un ataque con armas nucleares, con fines militares. Las ideas clave de su trabajo eran el uso de una red descentralizada con múltiples caminos entre dos puntos, y la división de mensajes completos en fragmentos que seguirían caminos distintos. Estas ideas conformarían una red que estaría capacitada para responder ante sus propios fallos e interferencias externas. En la misma época, Leonard Kleinrock ya trabajaba en el concepto de almacenar y reenviar mensajes en su tesis doctoral en el MIT.

Mezclando estos conceptos innovadores, cuatro centros de investigación independientes (DARPA, la corporación RAND, el MIT y NPL en el Reino Unido) acabarían convirtiéndose en los primeros nodos experimentales de ARPANET. Esto sería el germen de las comunicaciones entre computadores, y lo que hoy conocemos como redes de ordenadores, Internet incluido.

Por último, el desarrollo de microprocesadores poderosos y económicos, junto con la posibilidad de intercambiar información entre computadores, y el desarrollo de las redes locales de alta velocidad a principios de 1970 propició la creación de nuevos conceptos en el tratamiento y procesado de la información, en los que las comunicaciones no solo sirven para compartirla, si no también para compartir los recursos de los propios computadores, al mismo tiempo que se descentralizan las tareas. Esta tendencia se ha acelerado por el desarrollo de software para sistemas distribuidos, diseñado para soportar el desarrollo de aplicaciones distribuidas. Este software permite a los ordenadores coordinar sus actividades y compartir los recursos del sistema hardware, software y datos.

2.1.2. SISTEMAS DISTRIBUIDOS

En general, se consideran sistemas distribuidos a los sistemas en los que existen varios procesadores conectados entre sí, que trabajan de manera conjunta. Generalmente se comunican a través de una red, y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo, estableciéndose la comunicación mediante un protocolo prefijado previamente.

Las características más reseñables de un sistema distribuido son las siguientes:

- Concurrencia: Esta característica de los sistemas distribuidos permite que los recursos disponibles en la red puedan ser utilizados simultáneamente por los usuarios y/o agentes que interactúan en la red.
- Fallos independientes de los componentes: Cada componente del sistema puede fallar independientemente, con lo cual los demás pueden continuar ejecutando sus acciones. Esto permite el logro de las tareas con mayor efectividad, pues el sistema en su conjunto continua trabajando.
- Transparencia: El uso de múltiples procesadores y el acceso remoto debe de ser transparente al usuario.
- Compatibilidad entre los dispositivos conectados.

Las ventajas de tener un sistema distribuido, son múltiples. Herb Grosch formuló la que se llamaría "Ley de Grosch": "El poder de cómputo de una *cpu* es proporcional al cuadrado de su precio". Dicho enunciado sugería que la solución más eficaz en cuanto a costo es limitarse a un gran número de *cpu* baratos reunidos en un mismo sistema, con lo cual era una poderosa razón para la tendencia hacia la descentralización de las tareas. Pero la razón económica no es la única ventaja de los sistemas distribuidos:

- Los sistemas distribuidos generalmente tienen en potencia una proporción precio / desempeño mucho mejor que la de un único sistema centralizado.
- Satisfacen la necesidad de muchos usuarios de compartir ciertos datos, como datos personales, ventas de entradas o billetes de las líneas aéreas, etc.
- Con los sistemas distribuidos se pueden compartir otros recursos como programas y periféricos costosos, como pueden ser impresoras láser.
- Otra importante razón es lograr una mejor comunicación entre las personas, como puede ser el caso del correo electrónico, edición documentos compartidos, comunicación en tiempo real...
- Una flexibilidad mayor al tener una mejor distribución de la carga de trabajo entre las máquinas disponibles.

Las principales desventajas de los sistemas distribuidos, son la dificultad de desarrollo del software relativo a distribuir tareas, y la mayor complejidad de mantener una infraestructura más compleja, compuesta no solamente por los propios

ordenadores, si no también por redes, protocolos de comunicación, personas encargadas de su mantenimiento, y todos los nuevos problemas derivados de estos elementos. Hace unos años también se habría podido citar la “capacidad de las redes de comunicación”, pero hoy en día, con el espectacular crecimiento de las mismas en los últimos años, por lo general no suele ser un problema.

2.1.3. CLASIFICACIÓN DE LOS SISTEMAS DISTRIBUIDOS

Existen diversos esquemas de clasificación para los sistemas de cómputo con varias cpu. Uno de los mas conocidos es la "Taxonomía de Flynn". Las cuatro clasificaciones definidas por Flynn se basan en el número de instrucciones concurrentes (control) y en los flujos de datos disponibles en la arquitectura:

- Una instrucción, un dato (SISD): Computador secuencial que no explota el paralelismo en las instrucciones ni en flujos de datos. Ejemplos de arquitecturas SIS son las máquinas con uni-procesador o monoprocesador tradicionales como el PC o los antiguos mainframe.
- Múltiples instrucciones, un dato (MISD): Poco común debido al hecho de que la efectividad de los múltiples flujos de instrucciones suele precisar de múltiples flujos de datos (en la práctica no se usa). Sin embargo, este tipo se usa en situaciones de paralelismo redundante, como por ejemplo en navegación aérea, donde se necesitan varios sistemas de respaldo en caso de que uno falle.
- Una instrucción, múltiples datos (SIMD): Un computador que explota varios flujos de datos dentro de un único flujo de instrucciones para realizar operaciones que pueden ser paralelizadas de manera natural. Por ejemplo, un procesador vectorial. Son útiles para los cómputos que repiten los mismos cálculos en varios conjuntos de datos.
- Múltiples instrucciones, múltiples datos (MIMD): Varios procesadores autónomos que ejecutan simultáneamente instrucciones diferentes sobre datos diferentes. Los sistemas distribuidos suelen clasificarse como arquitecturas MIMD; bien sea explotando un único espacio compartido de memoria, o uno distribuido. La práctica totalidad de sistemas distribuidos son de este tipo.

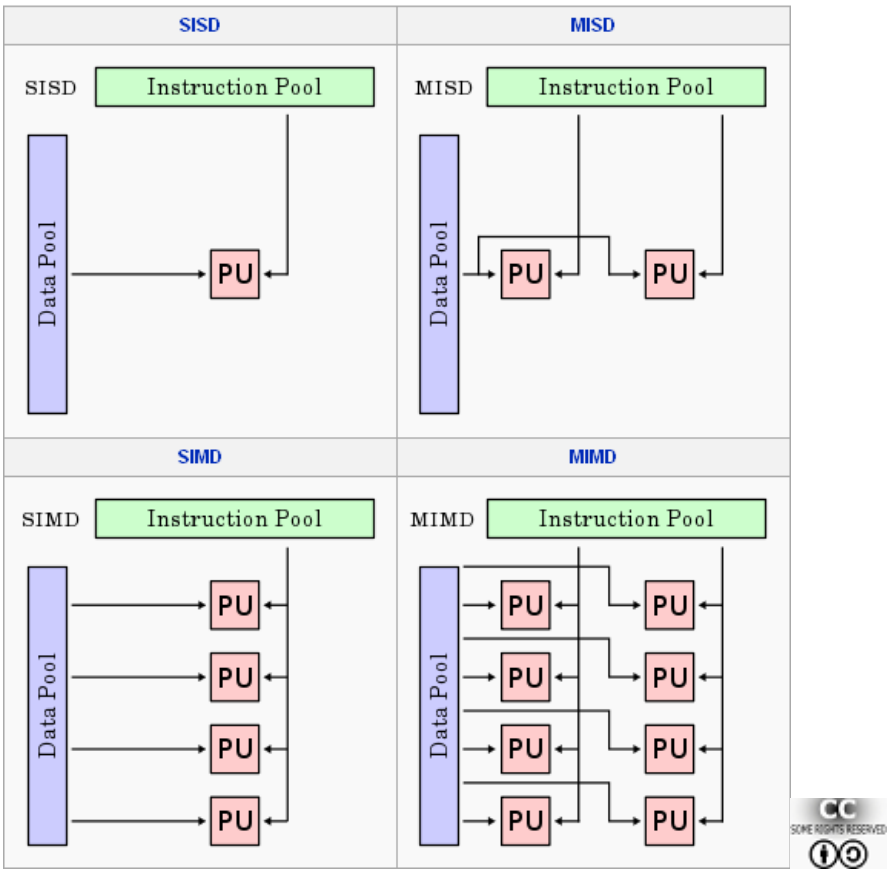


Figura 2.1.a: Esquema de las clasificaciones basadas en la Taxonomía de Flynn

Un subclasificación sobre las categorías definidas en la Taxonomía de Flynn incluye la división de las computadoras MIMD en dos grupos:

- Multiprocesadores: Poseen memoria compartida. Los distintos procesadores comparten el mismo espacio de direcciones virtuales.
- Multicomputadores: No poseen memoria compartida. Por ejemplo un grupo de PC conectados mediante una red.

2.1.4. SISTEMAS DE ARCHIVOS DISTRIBUIDOS

Generalmente, un sistema de archivos distribuidos consta de dos componentes muy distintos entre sí: el servicio de archivos y el servicio de directorios.

Un archivo es una secuencia de bytes sin interpretación alguna. Esto quiere decir que el contenido y estructura de un archivos es interpretado por el software de aplicación mas no por el sistema operativo sobre el que se está trabajando. Un archivo se caracteriza por tener atributos, tales como: el propietario, el tamaño, la fecha de creación y el permiso de acceso.

La utilidad del servicio de archivos consiste en proporcionar una adecuada administración de los atributos, definidos por el usuario, que estas poseen. Lo más común es encontrar algunos sistemas avanzados que permitan modificarlos después de sus creación, pero en algunos sistemas distribuidos las únicas operaciones que pueden realizarse sobre un archivo la creación, y la lectura. Es decir, una vez creado el archivo no puede modificarse. A este tipo de archivos se les denomina archivos inmutables.

Existen diferentes formas de utilizar los archivos, pero también existen dos formas de medir el grado de utilización de cada uso que se le puede dar a un archivo. Estas mediciones pueden ser mediciones estáticas, si se verifica el sistema del estado en un momento concreto de tiempo, o mediciones dinámicas, en las cuales se guarda un registro de todas las acciones para analizarlas posteriormente.

La estructura de un sistema es determinante para el servicio de archivos y directorios, para eso se debe diferenciar entre quiénes son los clientes y quiénes son los servidores. En algunos sistemas el servidor solamente puede actuar como servidor y el cliente solamente como cliente. Esto puede tener sus ventajas y desventajas puesto si en algún momento el servidor falla, entonces todo el sistema se paralizaría. En otros sistemas, sin embargo, el servidor de archivos y el de directorios son solamente programas del usuario, de esta manera se puede configurar el sistema para que ejecute o no el software de cliente o servidor en la misma máquina, como se desee. También hay sistemas mixtos, en los que un cliente hace de servidor en ciertos casos según necesidades (por ejemplo, el conocido "eMule").

En general, los sistemas de archivos distribuidos proporcionan la réplica de archivos como servicio a sus clientes. Es decir, se dispone de varias copias de algunos archivos, donde cada copia está en un servidor de archivos independiente. Este servicio de réplica se brinda por diversas razones, como pueden ser:

- Aumentar la confiabilidad al disponer de respaldos independientes de cada archivo. Es decir, si un servidor falla o se pierde permanentemente, no se pierden los datos.
- Permitir el acceso al archivo aunque falle un servidor de archivos. Si un servidor falla esto no debe ocasionar que el sistema se detenga.
- Repartir la carga de trabajo entre varios servidores. Con varios archivos duplicados en dos o más servidores, se puede utilizar el que tenga menor carga.

2.2. P2P

2.2.1. INTRODUCCIÓN

Una red P2P (*peer-to-peer*, en inglés, que se traduciría de "par a par" o de "punto a punto") se refiere a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red.

Este modelo de red contrasta con el modelo cliente-servidor el cual se basa en una arquitectura monolítica donde no hay distribución de tareas entre sí, solo una simple comunicación entre un usuario y una terminal en donde el cliente y el servidor no pueden cambiar de roles. Dichas redes son útiles para diversos propósitos, pero se usan muy a menudo para compartir toda clase de archivos que contienen audio, video, texto, software y datos en cualquier formato digital.

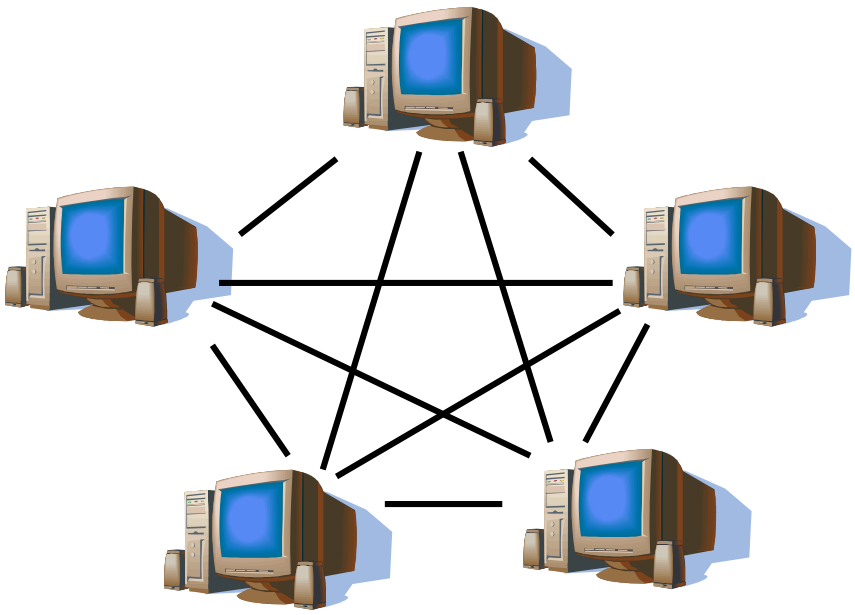


Figura 2.2.a: Ejemplo de una red basada en P2P

2.2.2. HISTORIA

La primera aplicación P2P fue Hotline Connect, desarrollada en 1996 para el sistema operativo Mac OS por el joven programador australiano Adam Hinkley. Hotline Connect, distribuido por Hotline Communications, pretendía ser una plataforma de distribución de archivos destinada a empresas y universidades, pero no tardó en servir de intercambio de archivos de casi todo tipo, especialmente de contenido ilegal y en gran parte pornográfico. Sin embargo, también se podían compartir archivos de contenido libre de distribución. El sistema Hotline Connect estaba descentralizado, puesto que no utilizaba servidores centrales, sino completamente autónomos: los archivos se almacenaban en los ordenadores de los usuarios que deseaban funcionar como servidores, y permitían, restringían o condicionaban la entrada al resto de usuarios, los clientes. En caso de que un servidor se cerrase, no existía ningún otro lugar del cual seguir descargando ese mismo archivo, y no quedaba más remedio que cancelar la descarga y empezar de cero en otro servidor.

Este sistema, en el que cada usuario dependía de un único servidor, no tardó en quedar obsoleto. Por otra parte, al ser una aplicación desarrollada fundamentalmente para una plataforma minoritaria como Mac OS no atrajo excesivamente la atención. Esto cambió con el nacimiento de Napster en 1999, a quien erróneamente se atribuye la invención del P2P. Aunque las transferencias de los archivos tenían lugar directamente entre dos equipos, Napster utilizaba servidores centrales para almacenar la lista de equipos y los archivos que proporcionaba cada uno, con lo que no era una aplicación perfectamente P2P. Aunque ya existían aplicaciones que permitían el intercambio de archivos entre los usuarios, como IRC o Usenet, Napster se presentó como la primera aplicación para PC especializada en los archivos de música mp3.

Después se estableció como líder P2P Audiogalaxy, otra aplicación centralizada de intercambio centrada en el contenido musical, que acabó siendo clausurada también por orden judicial. La RIAA (“Recording Industry Association of America”, la asociación estadounidense de discográficas) tomó estas resoluciones judiciales como victorias importantes encaminadas a acabar con la llamada "piratería", y emprendió una cruzada para acabar con ésta, que continúa a día de hoy con escaso éxito.

Acabar con las redes centralizadas era relativamente sencillo, pues bastaba con cerrar el servidor que almacena las listas de usuarios y archivos compartidos. Pero tras cada cierre de un servidor surgían nuevas aplicaciones, más modernas y seguras, aplicando nuevos conceptos en el mundo de la tecnología P2P. Particularmente interesante fue la creación de redes descentralizadas, que no dependen de un servidor central, y por tanto no tienen constancia de los archivos intercambiados.

En el año 2002, se dio un éxodo masivo de usuarios hacia las redes descentralizadas, como Kazaa, Grokster, Piolet, Morpheus, Ares, etc. La RIAA intentó, también por la vía judicial, acabar con los nuevos servicios descentralizados, y que permitían compartir varios tipos de archivos (no sólo mp3), pero Grokster y Morpheus ganaron sus juicios en abril de 2003.

Luego apareció "eDonkey 2000" (ya existía en el 2001 pero no era popular), aplicación que se mantuvo junto a Kazaa como líder del movimiento P2P. Más tarde, la aparición de otros clientes basados en el protocolo de eDonkey 2000, como Lphant, Shareaza, eMule causó el progresivo declive del programa original eDonkey 2000.

Otro paso importante lo marcó el protocolo BitTorrent, que pese a tener muchas similitudes con eDonkey 2000 proporciona, según los desarrolladores, una mayor velocidad de descarga, pero a costa de una menor variedad y longevidad de archivos en la red.

Los nuevos protocolos, cada vez más potentes, proporcionaban mayor velocidad de descarga, menor porcentaje de fallos, descentralización total (por ejemplo Kademlia, usado por eMule), e incluso anonimato. Pronto el uso de los P2P para descargar archivos se había convertido en una actividad cotidiana de la gente, al mismo tiempo que un verdadero quebradero de cabeza para la industria audiovisual.

Pero el potencial de la tecnología P2P no solo fue aprovechada para la descarga de contenidos audiovisuales. Numerosas ideas empezaron a ser desarrolladas usando esta nueva tecnología:

- Sistemas de archivos distribuidos, como Freenet, en desarrollo desde el año 2000. El objetivo de Freenet es almacenar documentos y permitir su acceso posterior por medio de una clave asociada, impidiendo que sea posible la

censura de los documentos y ofreciendo anonimato tanto al usuario que publica el documento como al que lo descarga.

- Sistemas de telefonía, como Skype, para realizar llamadas sobre Internet (VOIP) entre usuarios de la aplicación, incluso a teléfonos fijos.
- Cálculos científicos, en los que se requiere una gran capacidad de procesado, y se distribuye la carga de trabajo entre los usuarios de la red, como por ejemplo el conocido SETI@Home.
- Televisión P2P, que es una variante de la descarga de archivos, centrada en la descarga de video en tiempo real para poder ver canales de televisión de todo el mundo. Ejemplos podrían ser SopCast, Zattoo, TVants…

2.2.3. CARACTERÍSTICAS

Existen seis características deseables en toda red P2P:

- Escalabilidad. Las redes P2P tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos estén conectados a una red P2P mejor será su funcionamiento. Así, cuando los nodos llegan y comparten sus propios recursos, los recursos totales del sistema aumentan. Esto es diferente en una arquitectura del servidor-cliente con un sistema fijo de servidores, en los cuales la adición de más clientes podría significar una transferencia de datos más lenta para todos los usuarios.
- Robustez. La naturaleza distribuida de las redes peer-to-peer también incrementa la robustez en caso de haber fallos en la réplica excesiva de los datos hacia múltiples destinos, y (en sistemas P2P puros) permitiendo a los *peers* encontrar la información sin hacer peticiones a ningún servidor centralizado de indexado.
- Descentralización. Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red. En realidad, algunas redes comúnmente llamadas P2P no cumplen esta característica, como Napster, eDonkey2000 o BitTorrent.
- Los costes están repartidos entre los usuarios. Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.

- Anonimato. Es deseable que en estas redes quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo siempre que así lo necesiten los usuarios.
- Seguridad. Es una de las características deseables de las redes P2P menos implementada. Los objetivos de un P2P seguro serían identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje de las comunicaciones entre nodos, creación de grupos seguros de nodos dentro de la red, protección de los recursos de la red...

2.2.4. CLASIFICACIÓN

Una posible clasificación de las redes P2P pudiera ser acorde a su grado de centralización:

- Redes P2P centralizadas: Este tipo de red P2P se basa en una arquitectura monolítica donde todas las transacciones se hacen a través de un único servidor que sirve de punto de enlace entre dos nodos, y que a la vez almacena y distribuye los nodos donde se almacenan los contenidos. Poseen una administración muy dinámica y una disposición más permanente de contenido, sin embargo, está muy limitada en la privacidad de los usuarios y en la falta de escalabilidad de un sólo servidor, además de ofrecer problemas en puntos únicos de fallo, situaciones legales y enormes costos en el mantenimiento así como el consumo de ancho de banda. Algunos ejemplos de este tipo de redes son Napster y Audiogalaxy.
- Redes P2P "puras" o totalmente descentralizadas: las redes P2P de este tipo son las más comunes, siendo las más versátiles al no requerir de una gestión central de ningún tipo, lo que permite una reducción de la necesidad de usar un servidor central, por lo que se opta por los mismos usuarios como nodos de esas conexiones y también como almacenistas de esa información. En otras palabras, todas las comunicaciones son directamente de usuario a usuario con ayuda de un nodo (que es otro usuario) quien permite enlazar esas comunicaciones. Algunos ejemplos de una red P2P "pura" son Ares Galaxy, Gnutella, Freenet y Gnutella2.
- Redes P2P híbridas, semi-centralizadas o mixtas: en este tipo de red, se puede observar la interacción entre un servidor central que sirve como *hub* y administra los recursos de banda ancha, enrutamientos y comunicación entre

nodos pero sin saber la identidad de cada nodo y sin almacenar información alguna, por lo que el servidor no comparte archivos de ningún tipo a ningún nodo. Algunos ejemplos de una red P2P híbrida son Bittorrent, eDonkey2000 y Direct Connect.

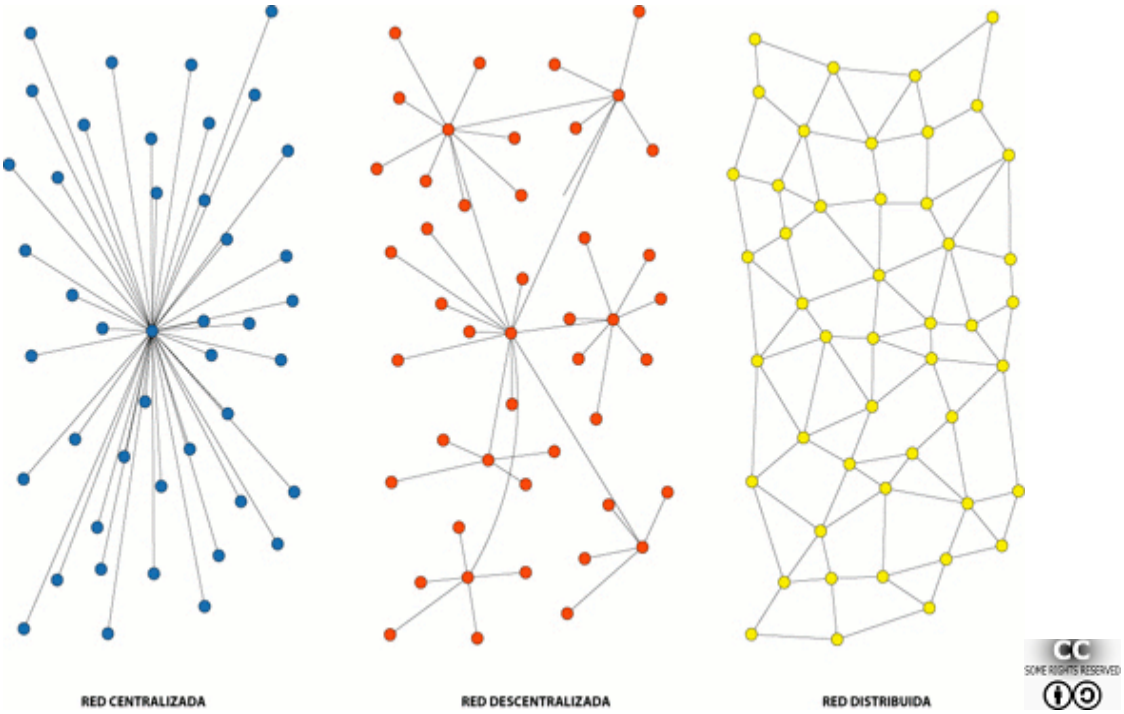


Figura 2.2.b: Las tres topologías de red según Paul Baran, que se aplican también al diseño P2P

Otras clasificaciones las redes P2P pueden ser en base a su estructuración, de acuerdo a su generación, acorde a sus características de anonimidad...

2.2.5. PROTOCOLOS Y REDES.

Debido a la popularidad de esta tecnología, y empujado por las necesidades y exigencias de los usuarios (y no solo en prestaciones, si no también en anonimato y seguridad), en la última década los protocolos e implementaciones de la tecnología P2P ha crecido a un ritmo vertiginoso, encontrándose casi tantas soluciones como problemas se han ido planteando en su desarrollo.

Existe una ingente número de aplicaciones P2P, y sus correspondientes redes y protocolos usados en las mismas. Podemos destacar algunas como la red creada por Ares. Esta aplicación fue creada a mediados de 2002 y desarrollado en lenguaje Delphi. Actualmente se encuentra disponible bajo licencia GPL. La red creada inicialmente por Ares la usan otras aplicaciones como FileCroc, KCeasy, Warez P2P...

El programa tuvo gran aceptación entre los usuarios debido a la rapidez y flexibilidad del protocolo (que permite búsquedas múltiples, colas de esperas remotas muy cortas y gran velocidad de descarga), la posibilidad de creación de salas de conversación en red, facilidad de uso, entre otras cosas. Su flexible algoritmo tiene como prioridad el poner en primer lugar en una cola remota de espera a usuarios que tienen menos porcentaje en una descarga. Esto permite que las colas remotas sean muy breves para aquellos usuarios que inician una descarga. Pero en archivos de gran tamaño, produce que la velocidad media de descarga se haga cada vez más lenta a medida que avanza la descarga. Además, actualmente soporta el protocolo BitTorrent, uno de los protocolos más usados, lo que es una ventaja añadida.

Ares usa un sistema DHT (Tablas de Hash Distribuido) para el manejo de fuentes. Las Tablas de Hash Distribuido son una clase de sistemas distribuidos descentralizados que proveen un servicio similar a las tablas hash, en las que hay un par "llave - valor", y con el cual los usuarios puede recuperar fácilmente el valor a partir de la llave. La responsabilidad de mantener la correcta correspondencia entre las llaves y sus valores corresponden a todos los nodos de la red, lo que hace que el nivel de corrupción de los valores sea mínimo aunque los participantes cambien. Esto permite a las tablas DHT escalar su cantidad de usuarios a valores extraordinariamente altos, manteniendo una elevada eficacia y una elevada tolerancia a fallos.

Otro conocido cliente es BitTorrent. El primer cliente basado en este protocolo, conocido como “BitTorrent”, fue creado por el programador Bram Cohen, en octubre de 2002. El protocolo BitTorrent fue desarrollado originalmente por él y está basado en software libre. Actualmente es innumerable la cantidad de aplicaciones distintas que usan este protocolo (se puede encontrar una lista de ellas en el siguiente enlace: http://es.wikipedia.org/wiki/Anexo:Clientes_BitTorrent). A diferencia de los sistemas de intercambio de archivos tradicionales, su principal objetivo es el proporcionar una forma eficiente de distribuir un mismo archivo a un

gran grupo de clientes, forzando a todos los que descargan un archivo a compartirlo también con otros. Primero se distribuye por medios convencionales un pequeño archivo con extensión ".torrent", el cual contiene la dirección de un "servidor de búsqueda", que se encarga de localizar posibles fuentes con el archivo o parte de él. Este servidor realmente se encuentra centralizado y provee estadísticas acerca del número de transferencias, el número de nodos con una copia completa del archivo y el número de nodos que poseen sólo una porción del mismo. El sistema se encarga de premiar a quienes compartan más, a mayor ancho de banda mayor el número de conexiones a nodos de descarga que se establecerán. Este método produce importantes mejoras en la velocidad de transferencia cuando muchos usuarios se conectan para bajar un mismo archivo.

El protocolo bitTorrent es particular tanto en su composición, como en su funcionamiento. En su composición participan elementos como "Peers" (los usuarios de la red), "Leechers" (usuarios descargando que no tienen el archivo completo), “Seeds” (usuarios con el archivo completo), "Tracker" (servidor especial que contiene la información necesaria para que los *peers* se conecten unos con otros). En cuanto a su funcionamiento, el protocolo BitTorrent se basa en algunos de los siguientes algoritmos:

- Rarest First Algorithm: Este algoritmo define la estrategia usada por el protocolo BitTorrent para seleccionar la siguiente pieza (porción de archivo) a descargar. Cada par mantiene una lista del número de copias de cada pieza en su conjunto de pares y usa esta información para definir su conjunto de las piezas más raras, las cuales son las primeras que se sirven si bajan de un determinado número mínimo.
- Choke Algorithm: Este algoritmo define la estrategia usada por el protocolo BitTorrent para seleccionar el siguiente *peer* con el que interactuar. Se usa para garantizar un buen ratio subida/bajada entre los *peers*, dando prioridad a los que poseen más velocidad de descarga, al mismo tiempo que posibilita que los clientes interesados en descargar obtengan rápido una primera porción del archivo, lo que les permite empezar a compartir, y por lo tanto aumentar al mismo tiempo sus posibilidades de descargar más porciones de archivos.
- DHT: Además, se han hecho varias mejoras adicionales al protocolo BitTorrent, como el uso de DHT, gracias al cual cada nodo de la red conserva información de los nodos vecinos. De esta forma se evita el cuellos de botella

del servidor, ya que si este se cae la totalidad de la información de los *peers* está todavía disponible en los propios clientes.

Otro de los clientes posiblemente más conocidos es eMule, evolución del desaparecido cliente eDonkey2000. Los dos clientes coexistieron un tiempo, pero la compañía propietaria de eDonkey2000, MetaMachine, alcanzó un acuerdo en 2006 con la RIAA para evitar un juicio por infracción de los derechos de propiedad intelectual. La compañía dejó de distribuir su software y acordó pagar una compensación de 30 millones de dólares. Pero eMule ya era tan o más popular que su antecesor, GPL, y mucho mejor preparado gracias a protocolos como Kademlia, que a día de hoy sigue funcionando con normalidad.

El protocolo eMule, a pesar de no ser el más eficiente en cuanto a velocidad, es uno de los más completos gracias a su licencia GPL y a los continuos aportes de los desarrollares. En cuanto a su funcionamiento podemos destacar:

- Ofuscación del protocolo: Sirve para evitar que las conexiones del eMule sean detectadas y bloqueadas por los ISP. La "Ofuscación de Protocolo" es una característica que hace que eMule esconda su protocolo al comunicarse con el servidor u otros clientes. Sin ofuscación, cada comunicación de eMule tiene una estructura predeterminada que puede ser fácilmente reconocida e identificada por un observador (no confundir "ofuscación de protocolo" con anonimato).
- Compartir fracciones de archivos: Los clientes pueden compartir fracciones de archivos aunque estos no estén completamente descargados
- Detección de errores: eMule utiliza algoritmos de detección de errores. De esta manera es casi imposible que se corrompan los archivos que se descargan. El sistema AICH (Advanced Intelligent Corruption Handling) utiliza el método de "hash de bloques" para fragmentar en trozos de archivo de 180 KB, disminuyendo muy notablemente la cantidad de datos que hay que volver a bajar para corregir un error de transmisión.
- Transferencias comprimidas: Cada vez que eMule transmite datos, los comprime con la librería *zlib* para ahorrar ancho de banda, de forma completamente transparente al usuario.
- Independencia de los nombres de archivo: En otros programas, cuando se renombra un archivo, éste ya no se considera el mismo. eMule genera un *hash*

para identificar el archivo, agrupando todos los archivos con el mismo *hash* como un mismo archivo con distintos nombres.

- Sistema de créditos y colas: Se recompensa a los usuarios que han subido más datos dándoles más prioridad a la hora de progresar dentro de la cola de espera. Los modificadores se calculan en base a la cantidad de datos transferidos entre dos clientes, el cual directamente afecta a la valoración de las peticiones de clientes y su posición en la cola.

En los últimos tiempos, la presión a los servidores eMule, la industria discográfica, operadores de comunicaciones, etc, ha propiciado un nuevo protocolo llamado Kademlia que crea una red del mismo nombre, bajo la que eMule funciona de manera conjunta junto con el original. La red "Kad" es una red totalmente descentralizada donde todos los nodos son iguales. Esto facilita que eMule pueda sobrevivir a una posible caída de la red de servidores.

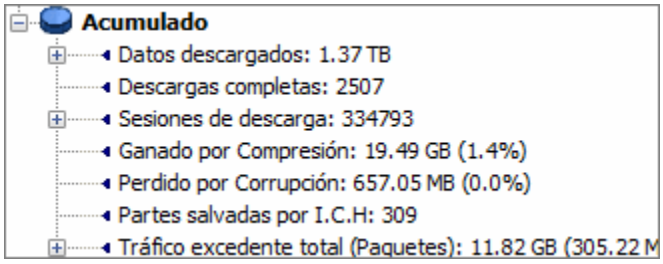


Figura 2.2.c: Estadísticas reales de uso de eMule. Se pueden observar elementos del protocolo como I.C.H, ganancia por compresión, etc.

La lista de protocolos, y la bibliografía existente sobre ellos es extensa, no centrándose únicamente en los programas de transmisión de archivos. Como hemos comentado anteriormente, existen múltiples aplicaciones que usan tecnología P2P, con sus propios protocolos, y sus propias redes. Telefonía VOIP, TVP2P, Sistemas de archivos distribuidos, etc. Podemos encontrar documentación en la referencia número.

2.3. .NET

2.3.1. DESCRIPCIÓN

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.

.NET ofrece una manera rápida y económica, a la vez que segura y robusta, de desarrollar soluciones, permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

El framework de .NET es la base principal del entorno creado para sistemas operativos de Miscrosoft. Incluye una enorme librería de soluciones codificadas que responden a la mayoría de las necesidades involucradas en el desarrolo de software. También ofrece el entorno de ejecución o CLR bajo el que corren todas las aplicaciones desarrolladas con este framework.

La librería de clases "base" provee una larga lista de características, incluyendo interfaz de usuario, datos, acceso a datos, conectividad con bases de datos, criptografía, desarrollo de aplicaciones web, algoritmos numéricos y comunicaciones de red. Estas librerías ofrecen soluciones fáciles de aplicar, que convinadas con el propio código desarrollo por los programadores resultan el entorno ideal de trabajo.

Los principales componentes del marco de trabajo en .NET son:

- El conjunto de lenguajes de programación
- La Biblioteca de Clases Base o BCL
- El Entorno Común de Ejecución para Lenguajes o CLR por sus siglas en inglés.

Debido a la publicación de la norma para la infraestructura común de lenguajes (CLI por sus siglas en inglés), el desarrollo de lenguajes se facilita, por lo que el

marco de trabajo .NET soporta decenas de lenguajes de programación, entre ellos Visual Basic. Net, en el que estará basado el desarrollo del proyecto.

2.3.2. PRINCIPALES CARACTERÍSTICAS

Dentro de las características que proporciona el framework de .net, y el entorno de trabajo podemos destacar las siguientes:

- Interoperabilidad: La interacción entre nuevos y antiguos desarrollos es comúnmente requerida. El framework de .NET provee tanto métodos de acceso a aplicaciones fuera del entorno .NET como posibilidades de interacción con antiguas aplicaciones desarrolladas en el mismo entorno. Por ejemplo, se puede acceder a componentes COM gracias a las funcionalidades que ofrecen los espacios de nombres System.Runtime.InteropServices y System.EnterpriseServices del framework.
- CLR: El CLR, o entorno común de ejecución es una máquina virtual que ofrece el entorno de ejecución de .NET. Todos los programas se ejecutan bajo la supervisión del CLR, garantizando ciertas propiedades, como áreas de memoria independientes, seguridad, control de excepciones, etc.
- CLI: Common Language Infrastructure (en español, Infraestructura de Lenguaje Común). El propósito del CLI es proveer un lenguaje neutral que sirva como plataforma para el desarrollo y ejecución de aplicaciones, incluyendo funciones para el control de excepciones, colector de "basura", seguridad e interoperabilidad. Distintos lenguajes podrán usar este lenguaje neutral para el desarrollo de sus aplicaciones, y funcionar en multiples plataformas hardware y software sin necesidad de recompilar el código.
- CTS: El framework de .NET introduce un Sistema Genérico de Tipos, que especifica todos los posibles tipos de datos y constructores de los mismos soportados por el CLR y como deben o no deben interactuar entre ellos. Gracias a esto, el entorno de .NET soporta intercambio de información entre programas escritos en cualquier lenguaje que use el framework de .NET.
- BCL: Es la Librería de Clases Base, que es parte de la Librería de Clases del Framework (FCL). Es una una librería de funcionalidades disponibles para todos los lenguajes que usen .NET. La BCL proveen clases que encapsulan las funciones comunes, incluyendo lectura y escritura de ficheros, renderizado gráfico, interacciones con la base de datos o manipulación de XML.

- Distribución simplificada: El entorno .NET incluye herramientas diseñadas para facilitar la distribución e instalación de aplicaciones, asegurándose de no interferir en el software previamente instalado.
- Seguridad: Control de posibles vulnerabilidades, como desbordamientos de buffer que son explotados por software malintencionado. Además, .NET provee modelos de seguridad extra para las aplicaciones desarrolladas bajo este entorno.
- Portabilidad: El diseño en .NET permite desarrollar software teoricamente independiente de la plataforma. Es decir, un programa que use el framework debería funcionar sin necesidad de nungun cambio en cualquier tipo de sistema para el cual haya sido diseñado el framework, como pueden ser Windows, Windows CE, XBOX 360, etc.
- Seguridad: .Net tiene sus propios mecanismos de seguridad, siendo el más conocido el llamado "CAS" (Code Access Security). CAS determina los permisos que se conceden para acceder al código y las rutinas, verificando en cada llamada los permisos necesarios, lanzando una excepción de seguridad en caso de que se no se disponga de dicho permiso. Incluso en la propia carga del ensamblado se verifican los permisos de acceso al mismo, y de este a los referenciados.

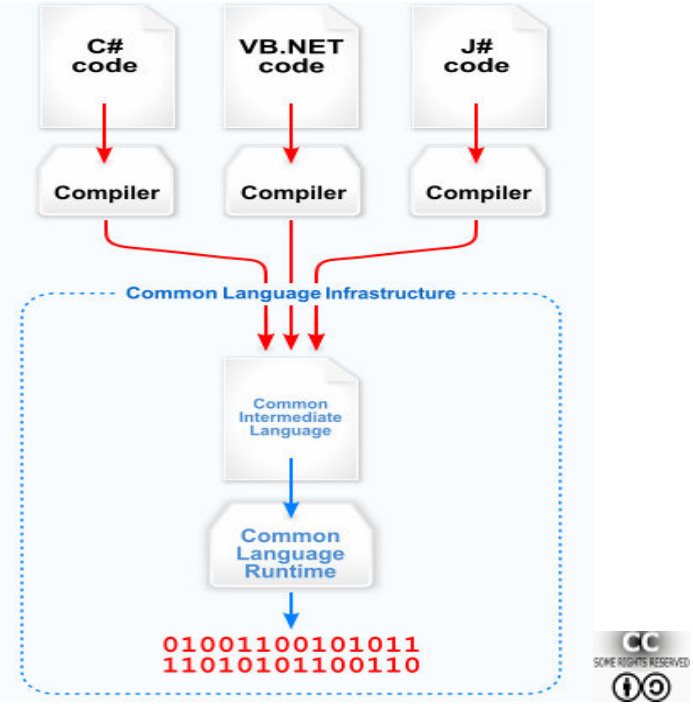


Figura 2.3.a: Ejemplo de interacción entre los elementos del framework para formar un compilado

2.3.3. El CLR

El CLR es el verdadero núcleo del *framework* de .NET, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET en un código intermedio, el MSIL (*Microsoft Intermediate Lenguaje*).

Para ejecutarse se necesita un segundo paso, un compilador "JIT" (*Just-In-Time*) es el que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .NET independencia de la plataforma de hardware. La compilación JIT la realiza el CLR a medida que el programa invoca métodos.

Figura 2.3.b: Estructura interna del entorno de ejecución en lenguaje común (CLR)

Además, .NET posee una enorme biblioteca de clases, perfectamente organizada y categorizada que aporta soluciones a prácticamente cualquier necesidad de los desarrolladores. Toda esta biblioteca de clases está totalmente documentada en la página de MSDN de Microsoft, y en español. Dicha página web contiene una gran cantidad de información técnica de programación, incluidos código de ejemplo en múltiples lenguajes, documentación, artículos técnicos y guías de referencia.

Esto, unido al entorno de desarrollo de .NET, Visual Studio 2005 que usaremos en el proyecto, proporcionan un marco inmejorable para el desarrollo de aplicaciones de todo tipo.

3. ANÁLISIS

3.1. VISIÓN GENERAL DEL *SOFTWARE*

3.1.1. INTRODUCCIÓN.

Lo primero que necesitamos es definir la composición general del producto que deseamos desarrollar. Ya hemos hablado sobre el marco general en el que se asientan los distintos elementos que componen el proyecto. Ahora, centrándonos en nuestra idea del producto, necesitamos identificar los distintos elementos que compondrán el software.

El objetivo principal es desarrollar un sistema distribuido de archivos mediante P2P. Este software debe proveer las funciones básicas de manejo de archivos: replicar tus archivos personales en otros nodos de la red, recuperar esos archivos replicados, y eliminarlos según se requiera.

En este proceso participarán los clientes, que son los que comparten su espacio físico, para que otros clientes puedan replicar sus archivos personales en él. Para esto necesitamos que una cierta parte de la capacidad de almacenamiento de este cliente esté reservada para el uso de de archivos P2P. Más adelante definiremos de forma más precisa la forma en la que se reserva este espacio, a la que llamaremos "trozos", que serán archivos de tamaño fijo que ocupan el espacio físico del cliente sin información relevante, o que almacenan dentro fragmentos de las replicas de archivos de otros usuarios.

Para administrar la red necesitamos un servidor, que se encarga de las funciones básicas de la red. Se necesita que el servidor sea capaz de administrar la red P2P, admitiendo usuarios y poniéndolos en comunicación. Además, por las características del sistema, en el cual replicas archivos en el espacio compartido de otros clientes, sin que estos sepan realmente si es de verdad un archivo de otro cliente o no, se necesita conocer que archivos están replicados y dónde.

Por último se necesita todo un sistema de comunicación compuesto por protocolos de comunicación entre clientes, entre clientes y servidor, seguridad, cifrado de archivos para que solo su legítimo dueño pueda tener acceso a ellos, etc.

3.1.2. FUNCIONES DE CLIENTE Y SERVIDOR

Ahora que tenemos una idea global de los elementos que compondrán el sistema, necesitados definir las competencias y funcionalidades de los distintos elementos. Para el funcionamiento de la red, los "clientes" deben ser capaz de conectar con un servidor central, que realiza diversas funciones:

- Validar la creación de nuevos usuarios, y la eliminación de los ya existentes.
- Conexión/desconexión de usuarios a la red.
- Almacenar datos de usuario.
- Interconexión de usuarios.
- Gestión del espacio disponible formado por la red de usuarios.

Los clientes, por su parte, deben ser capaz de realizar las siguientes funciones:

- Creación de nuevos usuarios, y eliminación de los existentes.
- Conexión/desconexión a la red P2P creada por el servidor, es decir, conexión al servidor, y a otros clientes.
- Reservar un determinado espacio de disco duro para "compartir" en la red P2P.
- Replicar los archivos personales elegidos en otros clientes, mediante el fraccionamiento y cifrado de los mismos.
- Aceptar "trozos" de archivos provenientes de otros usuarios, y ubicarlos en el "espacio compartido" de su disco duro.
- Mandar "trozos" de archivos contenidos en el "espacio compartido" a petición de otros clientes.
- Listar los archivos personales replicados en la red.
- Borrar los archivos personales replicados en la red.
- Guardar datos de configuración de usuario, en caso de ser requerido.

3.1.3. FLUJO DE MENSAJES

Una vez concretadas las funciones de cliente y servidor, necesitamos definir el flujo de mensajes que se produce a través de la red para hacer peticiones entre los participantes de la misma, mandar fragmentos de archivos, autentificación de usuarios, etc. La siguiente figura es un esquema general del flujo de mensajes entre cliente y servidor, y entre clientes. Más adelante se especificará cada uno de estos casos.

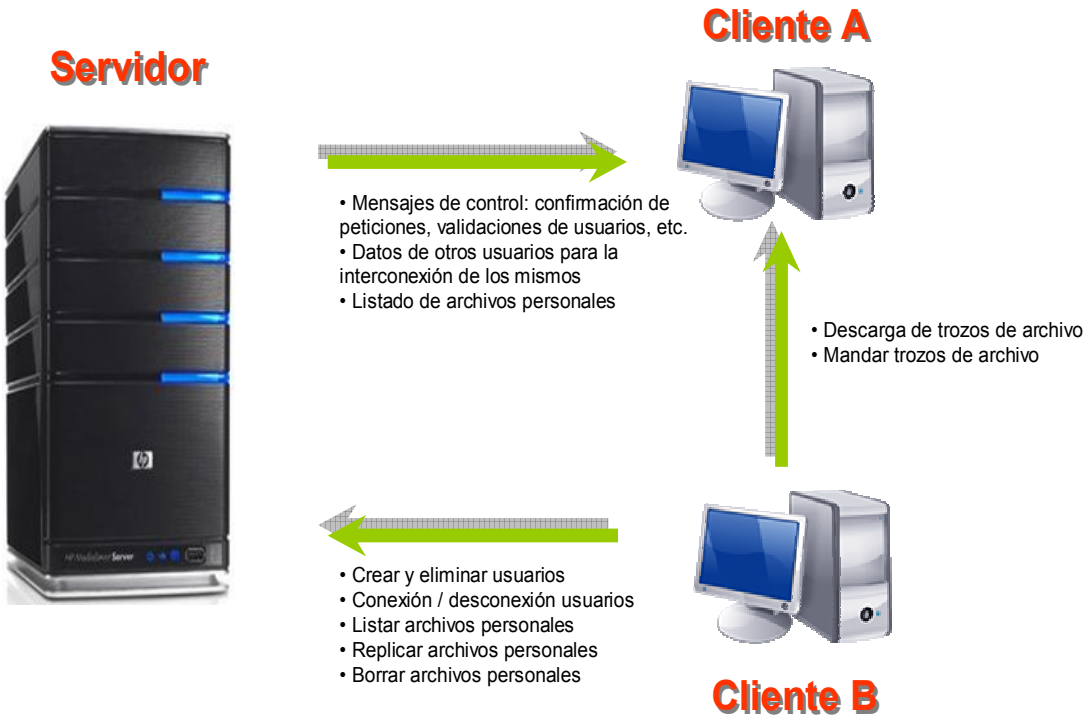


Figura 3.1.a: Flujo de mensajes entre clientes, y entre cliente y servidor.

3.2. ANÁLISIS DE LA COMUNUCACIÓN

3.2.1. PROTOCOLO DE COMUNICACIÓN.

Como se señaló anteriormente, una de las características principales de la tecnología P2P es que conforma una red de "nodos", para realizar sus tareas, ya se compartir espacio, descargar archivos, distribuir tareas, etc. Y por tanto, dichos nodos necesitan un protocolo de comunicación, tanto un protocolo de red para transmitir los mensajes, como un protocolo propio para que la información transmitida sea interpretada por los distintos nodos. El protocolo de comunicación, tanto entre el servidor y los clientes, como entre clientes es vía "TCP".

Este protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. Estos son dos rasgos fundamentales del protocolo TCP, ya que garantiza la integridad de los datos (corrección y completitud), algo fundamental sobre todo si los datos van cifrados, ya que es más complicado saber si la integridad de los datos recibidos es la correcta al no saber que se espera recibir. Aún así, para asegurar a corrección de los datos enviados, el formato de los "trozos" de archivo llevará integrado un pequeño "hash" del mismo que complementará este rasgo del protocolo, como más tarde explicaremos. El protocolo TCP también proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto "Puerto", gracias a lo cual será posible ejecutar varias instancias de programa en el mismo ordenador en el caso de requerirse.

El cliente deberá saber la dirección "IP" y el "puerto" de conexión con el servidor, o en su defecto un dominio que redirija a él. Además, como todo cliente actúa como servidor de otros clientes, al conectarse con el servidor deberá proporcionarle un "puerto" de escucha, a través del cual otros clientes conectarán con él.

Por último, será responsabilidad del servidor y de los clientes asegurar que la conexión a través de dichos puertos es posible, habilitando si fuera necesario canales de comunicación a través de *firewalls*, *proxys* y otros elementos similares.

3.2.2. FORMATO DE LA COMUNICACIÓN

Necesitamos, además de un protocolo de transmisión de mensajes, un protocolo interno para la interpretación de los mismos. Dependiendo de lo que se quiera transmitir, la composición de la información puede variar el tamaño, pudiéndose además recibir mensajes encadenados en una misma comunicación. Por lo tanto el formato de la comunicación diseñado debe ser capaz de distinguir entre mensajes distintos, tanto si se reciben varios en un mismo mensaje, como si se necesitan varios mensajes para recibir un solo. Definiremos el formato básico de la comunicación entre cliente y servidor, y entre clientes, es una cadena de *bytes*, con tres elementos: tamaño del mensaje total (que no tiene por qué coincidir con las dimensiones de los "paquetes de datos" recibidos a través del puerto de comunicación), orden o petición a ejecutar y datos de la petición, si los hubiera.

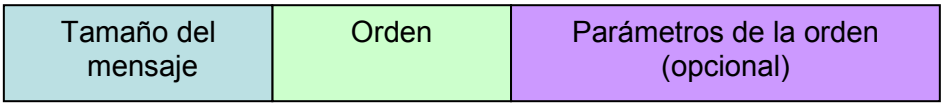


Figura 3.2.a: Formato de los mensajes transmitidos en la comunicación.

- Tamaño del mensaje: Este elemento es necesario para discernir entre varios fragmentos de un mismo mensaje que llegan de forma separada si este fuera muy grande, o varios mensajes distintos dentro de uno solo.
- Orden: Toda comunicación servidor cliente, o entre clientes, tiene un propósito específico y regulado. Este parámetro indica cual es el motivo de la comunicación.
- Parámetros de la orden: Todos aquellos datos que sirven para precisar el motivo de la comunicación y los datos de la misma.

3.3. ANÁLISIS DEL SISTEMA DE ARCHIVOS

3.3.1. FRACCIONADO DE ARCHIVOS

La idea básica del Sistema de Archivos Distribuido mediante P2P, es que podemos replicar nuestros archivos personales y guardar dichas replicas en los espacios compartidos de otros usuarios de la red.

Pero estos archivos pueden tener tamaños diversos, desde unos cuantos *Kbytes* hasta cientos de megas. Por otro lado, el espacio compartido por los usuarios es libre, un usuario puede compartir varios *GBytes* de espacio de su disco duro, pero también puede compartir solo unos pocos *MBytes*. Además, la replicación de archivos muy grandes en ordenadores remotos requeriría una conexión prolongada entre dos usuarios hasta que la transferencia estuviera completa, cosa que no siempre es posible.

De aquí surge la necesidad del fraccionamiento de los archivos en "trozos". Todos los "trozos" tendrían un tamaño suficientemente pequeño para poder transmitirle en cortos espacios de tiempo (siempre dependiendo de la capacidad de conexión), y a la vez un tamaño suficientemente significativo para que los datos de control (cabeceras) que identifiquen a estos trozos no supongan un tamaño proporcionalmente significativo respecto de los trozos.

El fraccionamiento de los archivos supone las siguientes ventajas:

- Porciones de un tamaño suficientemente reducido como para ser transmitidas en breves espacios de tiempo, lo que requiere conexiones poco prolongadas entre clientes.
- Mejor distribución proporcional en la red de los archivos replicados, al ser fraccionados y mandadas las partes de dicho archivo a distintos clientes, en vez de usar un único replicante para su almacenaje.
- Mayor facilidad de replicación, al ser archivos relativamente pequeños los resultantes del fraccionamiento, independientemente del tamaño del archivo replicado. Además, el espacio compartido de cada cliente no necesita ser, individualmente, tan grande como el archivo a replicar.

- Detección más simple de anomalías en los archivos al poseer todos el mismo tamaño. A esto ayudará un *hash* que poseerá cada "trozo".
- Tamaño relativamente grande como para que los datos de control de "trozos" no supongan una proporción relevante en la cantidad de datos.
- Mayor seguridad al estar fraccionada la información y no poseer enteramente su replica un mismo usuario. Además, todos los "trozos" de archivo irán cifrados para mayor seguridad.

Desventajas derivadas del fraccionamiento de archivos:

- Más carga en la red al tener que realizar los usuarios más conexiones con otros, tanto para transmitir las replicas de los trozos, como para recuperarlas.
- Mayor espacio ocupado de disco ante la necesidad de información de control de "trozos" ("cabeceras")
- Mayor carga de procesamiento, ante la necesidad del fraccionamiento de los trozos cuando se procede a su replica, o cuando una vez recuperados se necesita volver a unirlos.

A pesar de existir ciertos inconvenientes en el fraccionamiento de archivos, como los anteriormente citados, las ventajas superan ampliamente a las desventajas. De hecho sería inviable un sistema sin fraccionado de archivos, ya que tiene graves inconvenientes:

- Requeriría conexiones prolongadas entre usuarios, de varias horas dependiendo del tamaño de los archivos enviados; de hecho habría un riesgo de que, durante estas conexiones prolongadas, la desconexión de uno de los dos usuarios provoque que todos los datos mandados hasta el momento se hayan perdido.
- Si un usuario está desconectado, no están disponibles en la red todas las replicas de archivos enteros que posee.
- Si un usuario borra dichos archivos, o se da de baja del sistema se pierden los archivos enteros de los que poseía replica.
- El poseer la información completa de otro usuario podría suponer un problema de seguridad de la información para este.

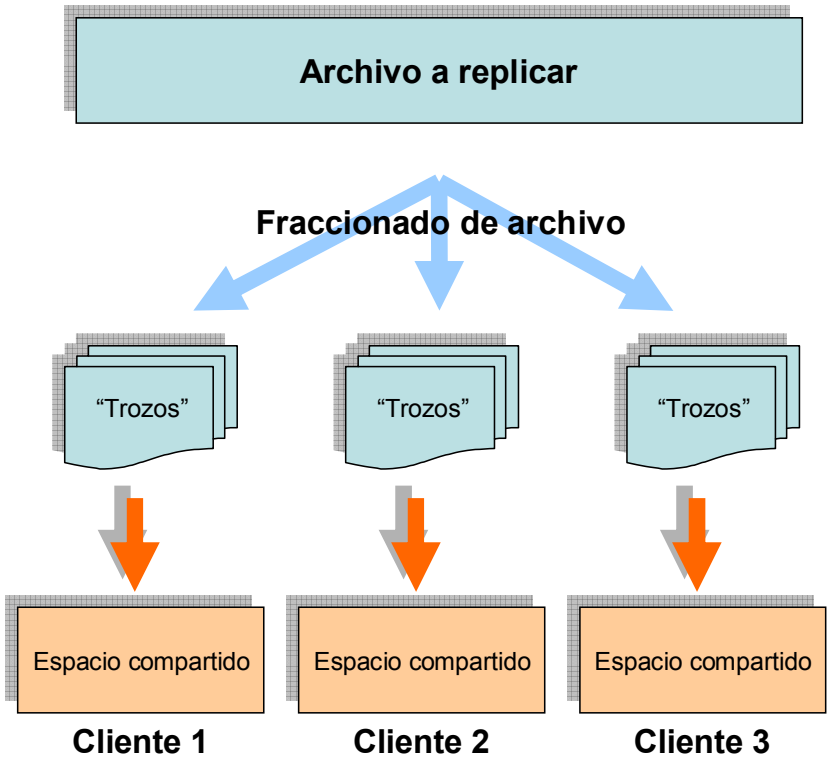


Figura 3.3.a.: Fraccionado de archivo en "trozos", para su réplica en distintos clientes

Así pues, el fraccionamiento de los archivos es beneficioso para el sistema, y necesario para su funcionamiento.

3.3.2. ESPACIO COMPARTIDO DE DISCO

En el Sistema de Archivos Distribuido mediante P2P se basa en compartir una porción de la capacidad de almacenamiento físico del usuario, gracias a la cual otros usuarios puedan hacer replicas de sus archivos personales en ese "espacio compartido", al mismo tiempo dicho cliente, como usuario de la red, puede replicar sus archivos en el espacio compartido de otros usuarios. La cantidad de "espacio compartido" es elegida de forma arbitraria por el usuario, pudiendo ser cero, aunque esto iría en contra de la filosofía de las redes P2P y el correcto funcionamiento de la red.

El software debe intentar garantizar que la cantidad de espacio de almacenamiento que el usuario desea compartir quede reservada para su acceso por parte de los demás clientes de la red P2P; otro proceso del sistema operativo, o el propio usuario podría intentar ocupar dicho espacio, pudiendo generar problemas al compartir menos espacio del que dice disponer. Para reservar este "espacio" la solución empleada es "ocupar" realmente ese espacio compartido con ficheros "huecos". Estos ficheros huecos no tendrán información útil, pero ocuparán el espacio reservado por el usuario, haciendo que a menos de que se borre manualmente, siempre esté disponible para que cuando otros usuarios hagan réplicas de sus ficheros, podamos ocupar este espacio compartido sustituyendo los archivos huecos por los "trozos" que nos manden. Los usuarios siempre podrían borrar manualmente dicho "espacio compartido", pero al ser archivos de dimensiones concretas siempre podemos detectarlo, y tomar las medidas oportunas, como la actualización en el servidor de la información referente al espacio que un usuario está realmente compartiendo. Dichos mecanismos se explicarán más adelante.

Por último, tenemos que darle un tamaño a los archivos que van a llenar dicho "espacio compartido". Una solución es darle exactamente el mismo tamaño y forma que los "trozos de archivo" (ficheros no huecos). De esta forma el usuario es incapaz de distinguir entre ficheros con información útil y ficheros con información no útil al tener las dos el mismo tamaño y estar cifrada la información útil. Esto añade un elemento más de seguridad al sistema de archivos, haciendo más compleja la explotación de la posible información fraccionada en los trozos.

3.3.3. CABECERAS Y *HASH* PARA EL CONTROL DE TROZOS

Tanto por el fraccionado de los archivos, como por el control de los mismos, necesitamos un sistema de control para toda la red P2P, un mecanismo para identificar los ficheros sin tener que transmitirlos enteros: unas cabeceras para los "trozos" que los identifiquen de forma única.

Características requeridas de las cabeceras:

- Tamaño pequeño respecto de los datos que representan. Un tamaño razonable sería cualquiera menor de un 1%.

- Identificar el trozo de forma inequívoca, incluso respecto del mismo trozo de un fichero idéntico pero de distinto usuario: fichero al que pertenece, posición dentro de ese fichero, numero de trozos, usuario, etc.
- Suficiente tamaño como para considerar despreciable la probabilidad de que el resultado de dos cabeceras cifradas distintas dé como resultado el mismo código de cabecera.

Otro elemento del sistema de archivos es un pequeño "hash" para el control de la integridad de los trozos. Si bien el protocolo TCP ya provee de un sistema que asegura en gran medida la integridad de los datos transmitidos, vamos a implementar un *hash* proporciona un control sobre la posible corrupción de datos en la red P2P, malintencionada o no, ya sea al transmitir los datos o al estar almacenados en los espacios físicos de los usuarios que componen la red, lo que complementa el control de fallos del propio protocolo TPC.

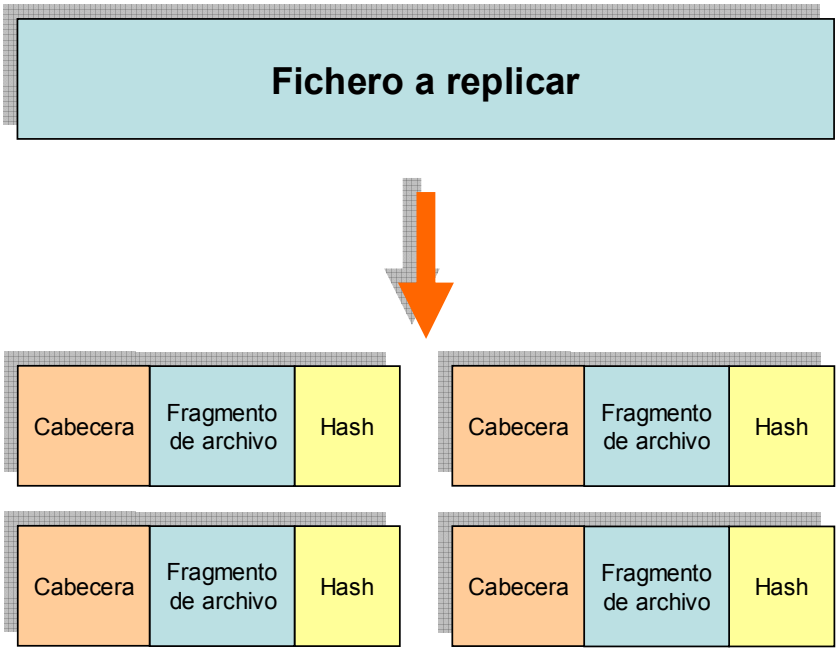


Figura 3.3.b.: Ejemplo de un fichero fraccionado, con los datos de control

3.3.4. FICHEROS DE DATOS QUE MANEJA EL CLIENTE

El cliente no tiene necesidad estricta de guardar ningún tipo de dato, lo cual beneficia en gran medida la portabilidad. Aún así, para comodidad de este usuario, el

programa ofrece la posibilidad de guardar sus "datos de usuario" para no tener que escribirlos cada vez que quiera hacer uso del software. Estos datos de usuario se guardarían en un "fichero de configuración", y constaría de datos tales como nombre de usuario, contraseña de conexión a la red, llave de cifrado, cantidad de "espacio compartido", puerto de escucha para conexiones entrantes, IP de conexión al servidor y puerto de conexión al servidor.

Por otro lado, si bien tampoco es imprescindible, se guardara un "log de errores" con las posibles excepciones y advertencias producidas durante la ejecución del programa, para su posterior análisis y solución de las posibles incidencias.

3.3.5. FICHEROS DE DATOS QUE MANEJA EL SERVIDOR

El servidor, a diferencia del cliente, si requiere del manejo de ciertos datos para el correcto funcionamiento de la red.

La principal tarea de el servidor es la gestión de usuarios, y para ello necesita conservar una lista de los mismos, para poder validarlos cuando se conecten y así discriminar posibles conexiones no validas y mantener un cierto control. Esta lista de usuarios debe contener al menos el nombre usuario, la contraseña de conexión, y la cantidad de espacio que comparte a la red. Otros datos interesantes podrían ser referentes a el número de replicas permitidas de sus archivos, o la fecha en la que realizo la última conexión a la red P2P, para la detección de usuarios "desaparecidos", considerando que los "trozos" que se almacenaban en su "espacio compartido" se han perdido, y poder hacer réplicas de los mismos con los trozos de otros usuarios que sí los posean.

Por la propia composición de la red y el tipo de software P2P que se pretende desarrollar, una de las características de seguridad implementadas en la red es el cifrado de archivos. Y debido a que los fragmentos de archivos se encuentran cifrados, y que estos poseen el mismo tamaño que los ficheros "huecos" existentes en el "espacio compartido" de usuario, el usuario no tiene conocimiento de cuánto de su espacio compartido son fragmentos de archivos con información válida, y cuáles son archivos "huecos". Así pues, es el servidor el encargado de conocer esta información; no tiene conocimiento de qué contienen los archivos, pues todos, incluso la cabecera, se encuentra cifrada con la "llave de cifrado" que únicamente conoce el usuario

propietario del archivo, pero mantiene una lista de usuarios en la que se relaciona "usuario – cabeceras de archivos no huecos" (el resto de los archivos serán archivos huecos por eliminación). Esta lista es útil para varios propósitos:

- Saber si un usuario ha borrado algún archivo "no hueco" manualmente, ya que cuando este se conecta, nos manda "la lista de cabeceras" de los archivos de su "espacio compartido".
- Si un usuario quiere reducir la cantidad de espacio compartido, saber qué trozos no debería borrar si no es necesario, y en caso de necesitarse, que trozos se han perdido.
- Si un usuario se borra o desaparece, saber qué trozos no están disponibles para tomar las medidas oportunas a este respecto.

Sabemos qué trozos son trozos con información válida, pero solo con esto no sabríamos quien es el dueño de los "trozos". Necesitamos esa información, ya que un cliente puede tener la necesidad de saber si están disponibles sus archivos personales (si están conectados al menos un usuario con una replicada de cada uno de los trozos de archivo), así como tener la posibilidad de recuperarlos. Pero el cliente no sabe dónde se hayan replicados dichos trozos, ni como conectarse a los usuarios que los tienen almacenados. Por otro lado el servidor sabe qué ficheros son "información" útil de cada usuario, pero no qué contienen ni de quién son, así que el servidor necesita dicha información para el funcionamiento de la red. A esta lista la llamaremos lista de "espacio compartido".

Para este fin usaremos una segunda lista, que estructuralmente es en parte una lista "invertida" de la primera, y contiene información de los usuarios, sus archivos personales replicados, y qué usuarios los tiene en su espacio compartido. Además, esta lista nos aporta la ventaja de que si queremos saber qué archivos tiene replicados un usuario en la red, al ser una lista invertida de la primera no hay que recorrer todos los usuarios comprobando su espacio compartido, basta con localizar el usuario dueño de los "trozos", y cotejar los replicantes de los mismos con la lista de usuarios conectados. A esta lista la llamaremos "lista de archivos personales".

Más adelante se precisará la estructura de estás dos listas de datos, las cuales deben ser consistentes la una con la otra.

3.4. DIAGRAMA DE CASOS DE USO

A continuación se describen las distintas interacciones entre los distintos tipos de usuario y el sistema mediante un diagrama de casos de uso, que permiten identificar los requisitos funcionales.

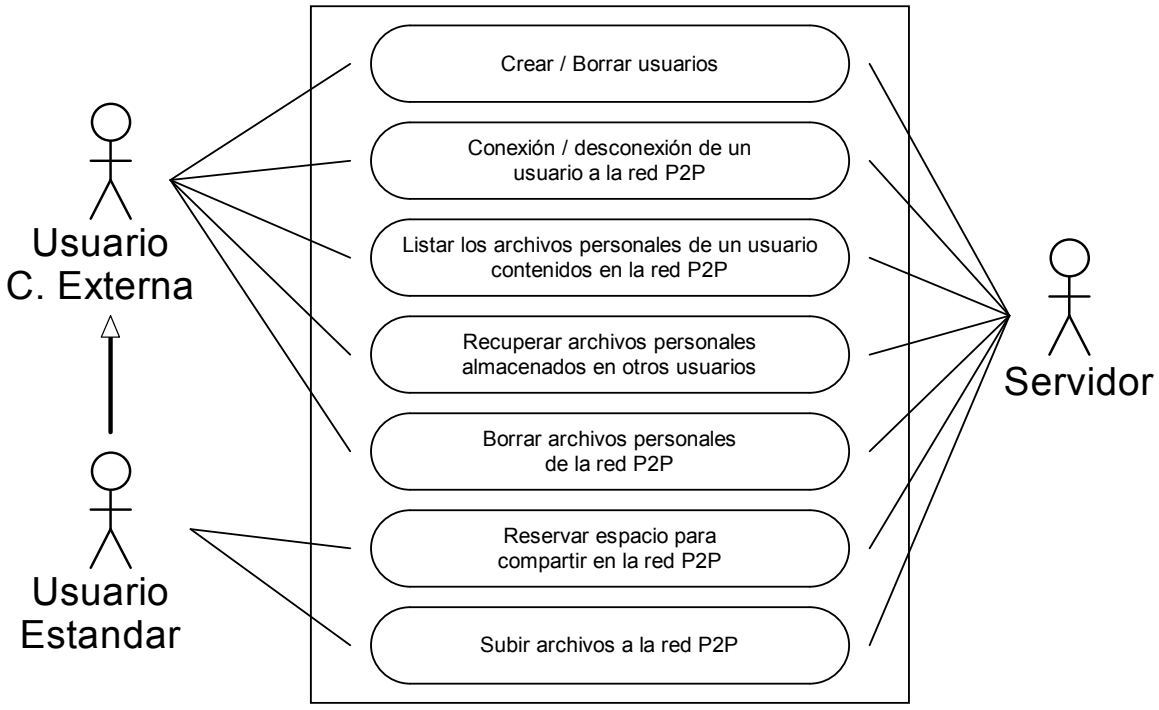


Figura 3.4.a: Diagrama de Casos de Uso

Usuario con conexión externa

Un usuario se puede conectar en "modo externo". Con esto le indica al servidor y al resto de nodos de la red que está conectado desde un ordenador distinto del habitual, y que por tanto no tiene "espacio compartido". Esto es necesario, ya que si no, cuando un usuario se conectara a la red P2P desde otro ordenador que no es el habitual, no tendría ahí su "espacio compartido", y el servidor pensaría que dicho usuario ha eliminado todos los "fragmentos" de información útil contenido en dicho espacio, con lo que los daría por perdidos.

Este usuario con “conexión externa” tiene acceso a todas las funciones las cuales no requieren tener “espacio compartido”: puede crear o borrar usuarios, puede conectarse o desconectarse a la red P2P, puede listar sus archivos personales replicados en la red, y recuperarlos si lo desea. También puede mandar borrar las replicadas sus propios archivos personales subidos a la red.

Usuario Estándar

Un usuario estándar es el usuario por defecto, aunque realmente es una derivación del usuario con “conexión externa”, ya que tiene todas sus funciones, y además, es capaz de realizar otras gracias a el “espacio compartido”: puede reservar una cantidad concreta de espacio en su disco duro para compartir en la red (así como variar la cantidad de espacio compartido si lo desea), es capaz de almacenar archivos provenientes de la red P2P en su espacio compartido (“trozos” que están replicando otros usuarios en su espacio compartido), y puede subir archivos a la red, ya sean replicas de sus archivos personales, o “trozos” de archivos del espacio compartido pedidos por otros usuarios.

Servidor

El servidor es un usuario especial, dentro de la red P2P. Este usuario es el encargado de la gestión de los usuarios (validar la creación de usuarios, aceptar conexiones a la red, etc), y de la gestión de la red en sí, encargándose de interconectar usuarios para la descarga o subida de archivos, teniendo en cuenta parámetros como la cantidad de espacio libre de cada uno para intentar la mejor distribución del mismo. Es un actor necesario para la consecución de todos los casos de uso, por lo que son casos de uso “colaborativos”.

3.5. ESPECIFICACIÓN DE LOS CASOS DE USO

Profundizando en los casos de uso, especificamos los requisitos identificados en el diagrama de casos de uso del apartado anterior

Nombre	**Crear / Borrar usuarios**
Actores	- Usuario con c. externa (y usuario estándar, por extensión) - Servidor
Objetivo	- Crear nuevos usuarios para acceder a la red P2P, o borrar los ya existentes
Precondiciones	- El servidor debe estar conectado
Poscondiciones	- Usuario eliminado (en el caso de "borrar usuarios") - Es posible que se produzcan replicas de trozos no huecos si se borraron (en el caso de "borrar usuarios")
Escenario Básico	**Crear usuario:** - El usuario se conecta al servidor, mandado nombre de usuario que quiere para su usuario, y la contraseña. - El servidor comprueba que ese usuario no existe, y lo valida o no, incluyéndolo en la lista de usuarios. - Si el usuario es validado, manda la cantidad de espacio que va a compartir a la red. - El usuario queda conectado. **Borrar Usuario:** - El usuario se conecta (nombre de usuario y contraseña correctos) - Manda una petición de dar de baja ese usuario - El servidor elimina toda referencia al usuario, tanto de su espacio compartido, como los archivos de los que es replicante. - El usuario borra su "espacio compartido" si lo tuviera, y todos los archivos de usuario. - El servidor manda replicar los "trozos" de archivo de otros usuarios que tuviera el usuario eliminado, y que por tanto se pierden.

Nombre	Conexión / Desconexión de un usuario a la red P2P
Actores	- Usuario con c. externa (y usuario estándar, por extensión) - Servidor
Objetivo	- Acceder a la red P2P con un usuario válido o desconectarse de ella, manejada por un servidor.
Precondiciones	- Conexión a la red: nombre de usuario y contraseña válidos. - Servidor Conectado - Desconexión de la red: usuario previamente conectado.
Poscondiciones	- Cliente conectado, o desconectado, según el caso.
Escenario Básico	**Conexión:** - El cliente manda los datos de conexión al servidor - El servidor valida esos datos y se lo comunica al cliente - El servidor comprueba las cabeceras del cliente y le valida como usuario conectado. **Desconexión:** - El cliente pulsa desconectar, se corta la comunicación, o se cierra por cualquier otra razón. - El servidor detecta esa desconexión, y lo elimina de la lista de clientes conectados.

Nombre	Listar archivos personales de un usuario
Actores	- Usuario con c. externa (y usuario estándar, por extensión) - Servidor
Objetivo	- Poder saber qué archivos se encuentran replicados en la red, así como cuantas replicas hay de los mismos (de sus trozos).
Precondiciones	- Usuario conectado a la red.
Poscondiciones	- Ninguna
Escenario Básico	- El usuario esta conectado al servidor. - Le pide listar sus archivos. - El servidor coteja los trozos del usuario, sus replicantes, y quienes están conectados. - Le manda una lista al usuario de los trozos y su disponibilidad. - El usuario recibe esa lista de cabeceras, la descifra, y sabiendo la disponibilidad de cada trozo, sabe la disponibilidad de cada archivo completo replicado.

Nombre	Recuperar archivos personales replicados
Actores	- Usuario con c. externa (y usuario estándar, por extensión) - Servidor
Objetivo	- Descargar de otros usuarios los fragmentos de archivos personales replicados en ellos.
Precondiciones	- Usuario conectado a la red. - Al menos una replica disponible de todos los fragmentos que componen el archivo. - Poder conectar con dichos usuarios (requerimiento externo al programa en ciertos casos).
Poscondiciones	- Ninguna.
Escenario Básico	- Se selecciona uno de los archivos del usuario. - Si tiene al menos una replica de cada trozo se manda al servidor la petición. - El servidor contesta con los datos para conectarse a los clientes con las replicas de nuestros archivos. - El cliente se conecta con los replicantes y descarga los fragmentos que componen el archivo - Si se ha conseguido descargar todos los trozos se descifran con la llave, se unen y se tiene el archivo

Nombre	Borrar archivos personales replicados
Actores	- Usuario con c. externa (y usuario estándar, por extensión) - Servidor
Objetivo	- Eliminar los “trozos” de los archivos personales replicados en otros usuarios.
Precondiciones	- Usuario conectado a la red.
Poscondiciones	- Archivo eliminado de la red.
Escenario Básico	- El usuario selecciona uno de sus archivos personales. - Le manda al servidor las cabeceras que componen el mismo. - El servidor borra dicho archivo de la lista de archivos personales del usuario. - El servidor borra el nombre de otros usuarios como replicantes de esos trozos.

Nombre	**Reservar espacio para compartir en la red**.
Actores	- Usuario estándar - Servidor
Objetivo	- Poder cambiar la cantidad de espacio compartido en la red.
Precondiciones	- Estar conectado a la red. - Si se quiere "aumentar" la cantidad de espacio compartido, se necesita disponer de esta cantidad de almacenamiento físico (requerimiento físico)
Poscondiciones	- El usuario reduce o aumenta su espacio compartido. - Es posible que se produzcan replicas de trozos no huecos si se borraron.
Escenario Básico	- El usuario, antes de conectarse, pone la nueva cantidad de espacio que desea compartir. - Si dicha cantidad es "superior", simplemente se crean trozos huecos en el directorio de espacio compartido, y al conectarse con el servidor, y mandarle las cabeceras, éste detecta el incremento de espacio. - Si se desea compartir menos, al conectarse al servidor se le manda la nueva cifra deseada. - El servidor comprueba si entre sus "trozos" de tu espacio compartido hay suficientes trozos "huecos" para borrar y ajustar ese tamaño compartido. - Si no hay suficientes "trozos huecos", añade a estos los trozos "no huecos" necesarios, y le manda la lista al cliente de cabeceras para que borre esos archivos. - Si tuvo que mandar borrar trozos no huecos, comunica con otros usuarios que poseyeran replicas también para su replicación.

Nombre	**Subir archivos a la red P2P**
Actores	- Usuario estándar - Servidor
Objetivo	- Poder subir fragmentos de archivo para que los almacenen otros usuarios, tanto si son fragmentos de archivos personales, como si son "trozos" almacenados en el espacio compartido.
Precondiciones	- Estar conectado a la red. - Ser capaz de escuchar peticiones de otros usuarios a través del puerto seleccionado si es para mandar "trozos" del espacio compartido (requerimiento externo al programa)
Poscondiciones	- Replica de archivo subido.
Escenario Básico	- El cliente (conectado) selecciona un archivo de su ordenador, calcula de cuantos "trozos" se compondría y se lo comunica al servidor. - El servidor calcula si hay espacio suficiente compartido en la red. - Si lo hay, pide a los usuarios que tienen ese espacio, que le manden cabeceras de "trozos" para comprobar si están vacías. - Una vez con suficientes cabeceras de "trozos" vacíos para contener el archivo, manda al usuario que quiere subir su archivo una lista que contiene los datos de conexión de los usuarios a los que se va a subir los archivos, y los "trozos" vacíos sobre los que se copiarán los trozos del archivo a replicar. - El cliente se conecta con dichos usuarios y sube los "trozos". - Si no se hubiera podido subir todos los trozos, se pide más espacio al servidor. - Una vez subidos todos los trozos, se manda al servidor una lista de los usuarios que las replicas de los archivos, para que actualice las listas de replicantes.

3.6. DIAGRAMAS DE ACTIVIDAD

Mediante los diagramas de actividad pasamos a describir el flujo de acciones de las distintas funcionalidades del sistema.

Diagrama de actividad de "Conectar Cliente":

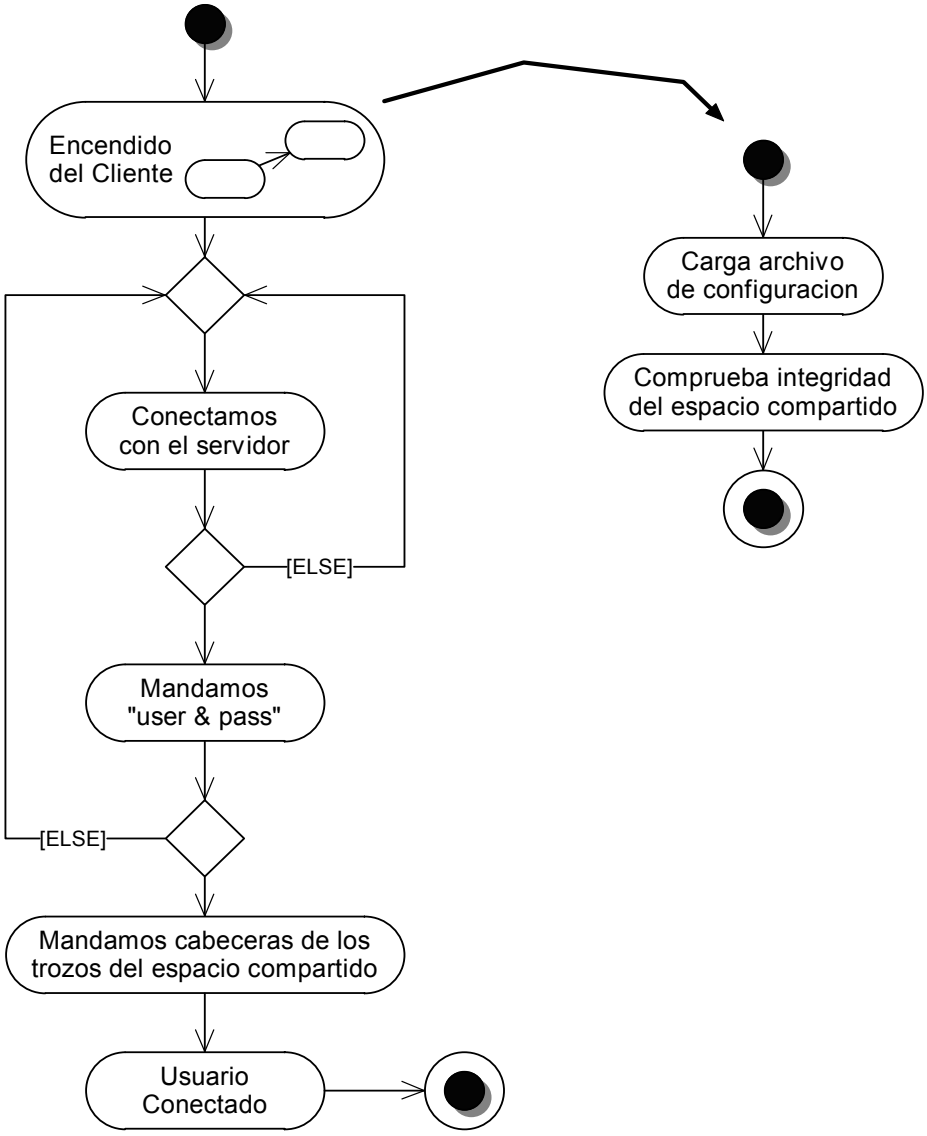


Figura 3.6.a: Diagrama de actividad de "conectar cliente" y subactividad de "encendido de cliente"

El diagrama de actividad de “conectar cliente”, muestra el proceso que sigue éste para conectarse al servidor. Hay que tener en cuenta que necesita la colaboración del servidor para validar los datos de la conexión, el nombre de usuario y contraseña. Es decir, que el servidor sería un agente “colaborativo”, tal y como se ha indicado en el diagrama de casos de uso.

El cliente enciende el programa. Si el cliente hubo guardado los datos de su configuración en una sesión anterior, dichos datos son recuperados del fichero “user.conf” y representados en el interfaz.

Por otro lado, si el cliente dispone de archivos en el directorio destinado a almacenar el “espacio compartido”, se producen una serie de validaciones que tienen como fin comprobar la integridad de estos archivos:

- Se comprueba el tamaño de los archivos y se borra aquellos que no coincidan en tamaño con el tamaño “de trozo”.
- Se crean ficheros si hiciera falta para completar el espacio que el usuario dice en su “configuración” que esta usando para su espacio compartido
- Se comprueba el “hash” de los archivos del espacio compartido por si hubiera alguno corrupto, y en dicho caso, se elimina.

Una vez realizadas todas estas comprobaciones, el cliente está listo para conectar con el servidor. Si consigue conectarse correctamente, el cliente le manda su nombre de usuario y contraseña para que le valide dentro de la red. Si es validado, el cliente le manda la lista de cabecera de los “trozos en su espacio compartido”, para que el servidor compruebe que el tamaño compartido coincida con el que debería tener, y que de esos trozos los “trozos no huecos”, que son replicas de archivos personales de otros usuarios, sean también los que deberían ser.

Se ha optado por no dibujar el diagrama de “desconectar” usuario, ya que carece de interés: el servidor detecta una desconexión del usuario mediante el “socket” por el que se encuentran conectados. Si detecta que se ha producido una desconexión, simplemente borra al usuario de la lista interna de usuarios conectados.

Diagrama de actividad de "Crear Usuario":

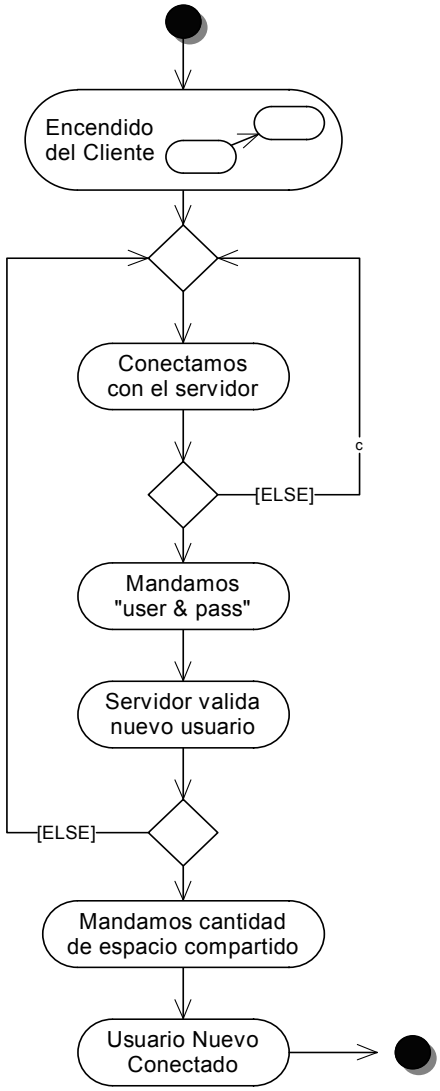


Figura 3.6.b: Diagrama de actividad de "crear usuario"

Como se puede observar, el proceso tiene cierta similitud con el de "conectar" al servidor: se inicia el cliente realizándose el mismo proceso descrito anteriormente, se introducen los datos de conexión del nuevo usuario, incluido el espacio compartido que queremos. El cliente conecta con el servidor, mandándole el nombre de usuario nuevo que desea crear, y la contraseña que tendría. El servidor comprueba que no exista ningún usuario con esos datos, y responde al cliente si acepta o no su solicitud. Si es aceptada el cliente manda un mensaje con la cantidad de espacio compartido; en este caso no manda las cabeceras, ya que si acaba de crear el usuario se entiende que

todas los trozos que tiene el espacio compartido de ese usuario son “huecos”. El cliente queda conectado.

Diagrama de actividad de “subir archivos a la red P2P”:

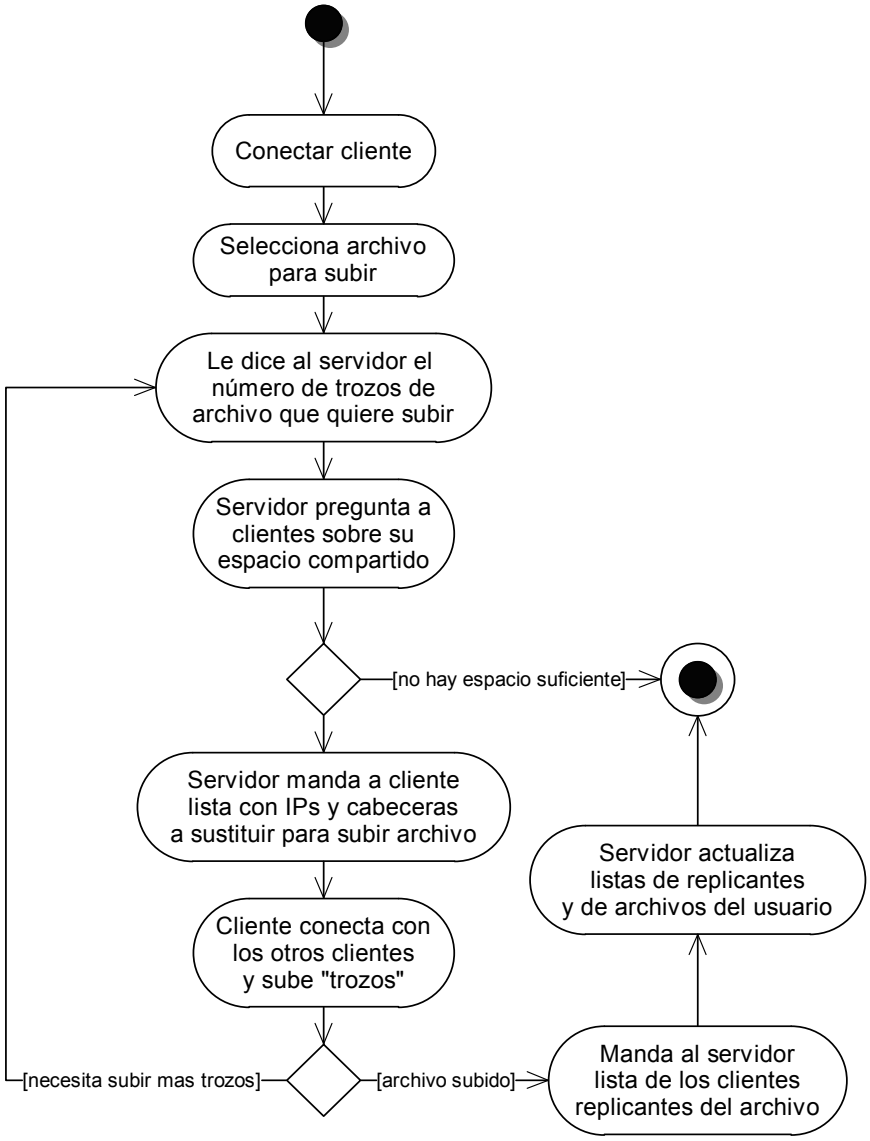


Figura 3.6.e: Diagrama de actividad de “subir archivos a la red P2P”

El cliente se conecta con el servidor. Una vez conectado, selecciona el archivo que quiere subir a la red, y le dice al servidor el número de trozos que necesita.

Para subir los trozos de los que se compone el archivo necesita mandárselos a otros clientes, y que estos lo sustituyan por "trozos vacíos" de su espacio compartido. Pero los clientes no saben cuales son trozos vacíos y cuales no en su espacio compartido. El servidor por su parte podría saber cuales son "trozos no vacíos" en los espacios compartidos de la gente, y necesitamos la información inversa, así que el servidor le pide a varios clientes que les manden un número concreto de cabeceras de trozos de su espacio compartido, y mira si estos son vacíos. Si lo fueran ya sabríamos qué trozos son vacíos y por tanto que trozos pueden ser sustituidos para la subida. En el caso de que alguno no sea vacío, le vuelve a pedir al cliente más cabeceras de trozos, así hasta conseguir el número de ficheros vacíos que va a requerir la subida.

Una vez el servidor tenga una lista de cabeceras de ficheros vacíos, le manda al cliente que quiere subir el fichero una lista de direcciones IPs junto con el puerto de escucha para conectar a estas direcciones, y la lista de ficheros vacíos que posee ese usuario por los que va a sustituir los trozos subidos.

El cliente va conectado con todos los clientes, codificando las partes del fichero y mandándolas, junto con la cabecera de los ficheros vacíos a sustituir en el cliente receptor de los trozos. Al mismo tiempo va llevando una lista de los clientes a los que ha conseguido subir los trozos y los que no; al final del proceso, es posible que no haya podido subir los trozos requeridos a todos los clientes, con lo que si faltan trozos por subir, vuelve a comunicar con el servidor pidiéndole más trozos. El servidor en previsión de esta situación ha guardado cual es el nombre de la última cabecera de trozo enviada por el cliente, con lo que le pide más cabeceras a partir de ésta.

Una vez terminado el proceso de subir las cabeceras a otros clientes, el cliente que ha subido el archivo manda al servidor una lista con los clientes a los que sí ha podido subir los archivos, y las cabeceras de los mismos. El servidor actualiza la lista de ficheros personales del usuario replicados en otros clientes, así como la lista de ficheros "no vacíos" de los usuarios replicantes.

Diagrama de actividad de "Borrar archivos personales replicados ":

Figura 3.6.h: Diagrama de actividad de "borrar archivos personales replicados"

El usuario está conectado. Lista sus archivos personales y decide que quiere borrar uno de ellos. Coge las cabeceras que componen ese archivo y se las manda al servidor indicándole que quiere borrar ese archivo de la red.

El servidor primero elimina esas cabeceras de la lista de trozos de usuario y sus replicantes. El archivo en si no se elimina de los clientes, pero como es igual que los archivos "vacíos" en la red y el servidor lo considera archivo vacío, funcionará como tal, pudiendo ser borrado o sustituido por otro archivo en cualquier momento.

Por otro lado el servidor también elimina las referencias del archivo de la lista de espacio compartido "no vació" de cada usuario, así que ese archivo, aunque realmente tiene información útil (y cifrada), pasará a ser considerado como archivo vacío a todos los efectos, lo que es lo mismo, ha sido borrado.

Diagrama de actividad de "borrar usuario":

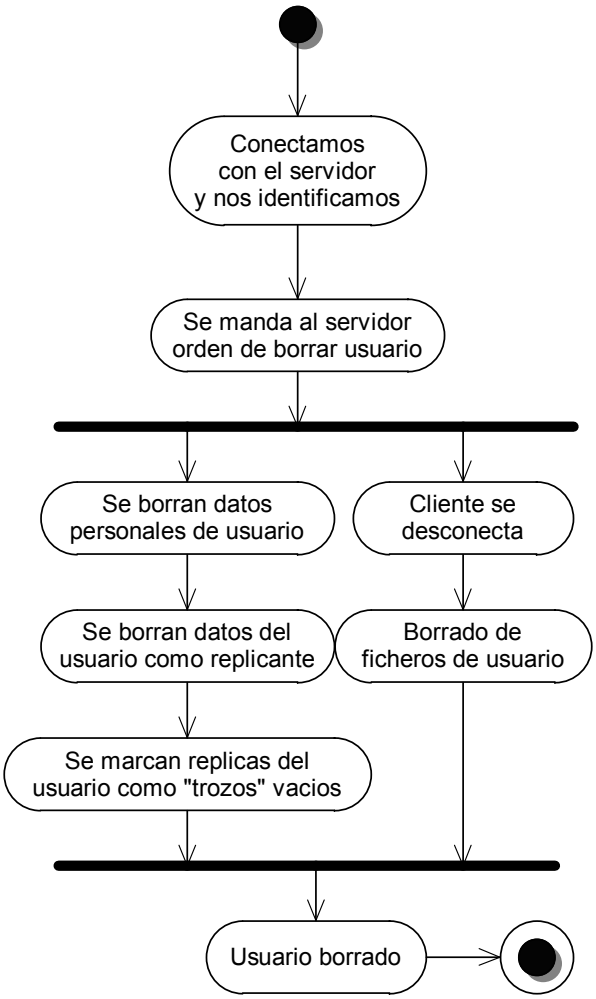


Figura 3.6.c: Diagrama de actividad de "borrar usuario"

En primer lugar nos conectamos al servidor, como si de una sesión normal de tratara; esta actividad ya ha sido descrita con anterioridad. Mandamos la orden de borrar usuario al servidor.

A partir de aquí, el usuario de desconecta y elimina los archivos de usuario que tuviera: ficheros de "espacio compartido" y el directorio en el que se guardan, si es que los tiene (puede que esté compartiendo espacio cero, o que se haya conectado como usuario externo); también borra el fichero de configuración de usuario, donde se guardan los datos mismo, y el "log" de programa.

Por otro lado el servidor elimina los datos personales del usuario: nombre de usuario, contraseña, espacio compartido, etc. Al ser borrado el usuario, los "trozos no huecos" que tuviera, replicas de ficheros de otros usuarios, se han perdido, con lo que tiene que borrar a este usuario como replicante de otros, y a su vez, si hiciera falta para mantener la integridad de los ficheros, mandar replicar más veces los "trozos" perdidos a otros usuarios que si los tengan replicados. Además, el usuario borrado es posible que tuviera ficheros personales replicados en otros usuarios, ficheros que ya no hace falta conservar, con lo que se sigue el proceso descrito anteriormente de "borrar archivo", pero nada todos los archivos replicados de ese usuario.

Diagrama de actividad de "cambiar el espacio compartido":

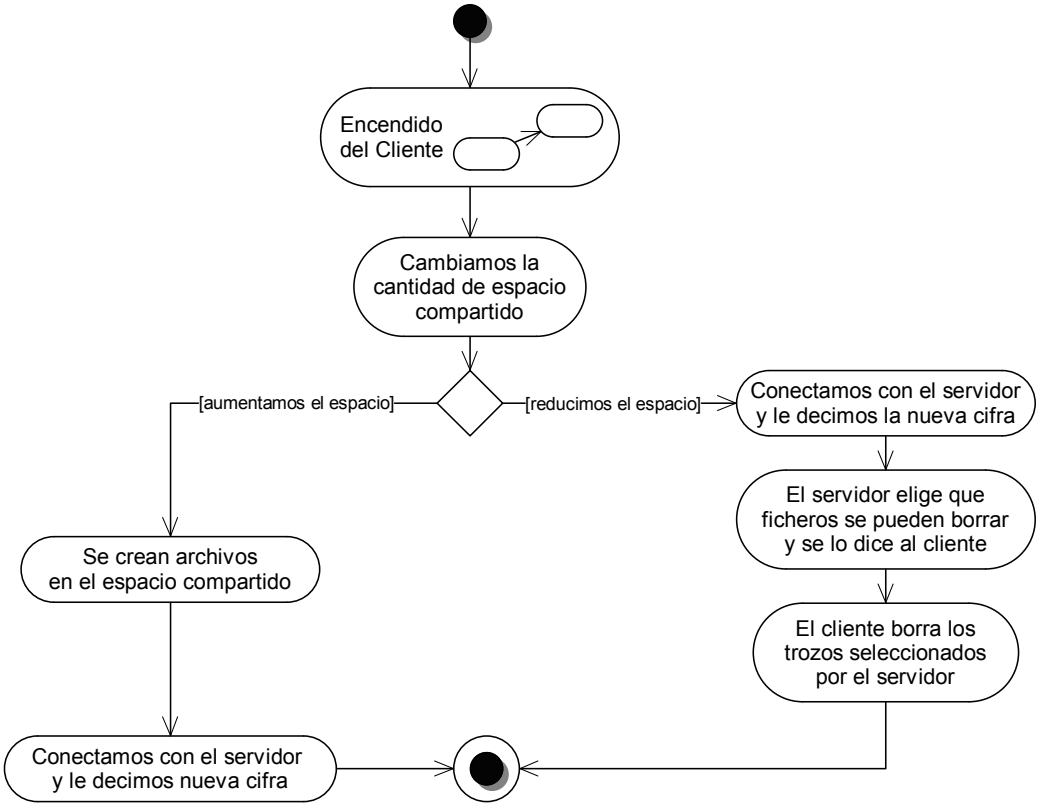


Figura 3.6.d: Diagrama de actividad de "cambiar el espacio compartido"

Primero encendemos el cliente, con las subrutinas que eso conlleva, y luego cambiamos la cifra del espacio compartido que deseamos tener.

Si aumentamos dicha cifra, se crean automáticamente “trozos” huecos hasta alcanzar la cifra seleccionada. Posteriormente al conectar con el servidor y mandarle las cabeceras, el servidor sabe que hemos aumentado nuestro espacio compartido ya que le estamos mandando más cabeceras de las normales, y actualiza los datos de cliente referentes al espacio que comparte con la red.

Si por el contrario lo que queremos es reducir la cantidad de espacio que compartimos en la red, no podemos borrar los trozos que queramos, ya que algunos pueden no ser trozos “huecos”, y ser replicas de los ficheros personales de otros usuarios. Así que si elegimos reducir nuestro espacio compartido, al conectarnos al servidor y mandar las cabeceras de los trozos, mandamos también la nueva cantidad de espacio compartido que queremos tener. El servidor analiza esas cabeceras. Primero intenta calcular si solo borrando los trozos “huecos” es posible reducir el espacio compartido del usuario hasta la nueva cifra que ha pedido. Si no es así, además de los trozos huecos selecciona los mínimos trozos “no huecos” posibles para alcanzar la cifra pedida por el usuario. Una vez confeccionada la lista de cabeceras de los archivos a borrar por el cliente, se la manda, y este los borra para ajustar su cantidad de espacio compartido.

Diagrama de actividad de "listar archivos de usuario":

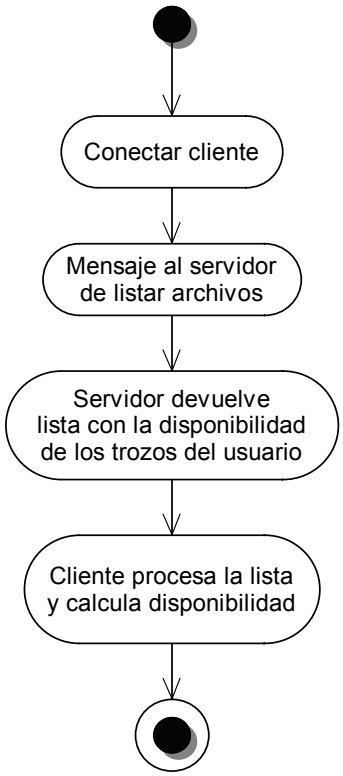


Figura 3.6.f: Diagrama de actividad de "listar archivos de usuario"

El cliente se conecta a la red P2P y le manda al servidor una petición para listar sus archivos personales replicados en otros clientes. El servidor realmente no sabe qué archivos ha replicado el cliente, solamente conoce las cabeceras de los trozos replicados, que van cifradas.

Así pues el servidor coteja la lista de trozos replicados del usuario, con la lista de usuarios conectados a la red, y le manda al cliente una lista de las cabeceras de los "trozos" y la disponibilidad de cada trozo.

El cliente recibe esa lista, descifra las cabeceras, y entonces sabe a qué archivo pertenece cada una, y la disponibilidad de cada trozo en la red en ese momento. Por lo tanto, sabe la disponibilidad del archivo que forman todos los trozos de los que se compone y se la muestra al cliente.

Diagrama de actividad de "recuperar archivos personales replicados":

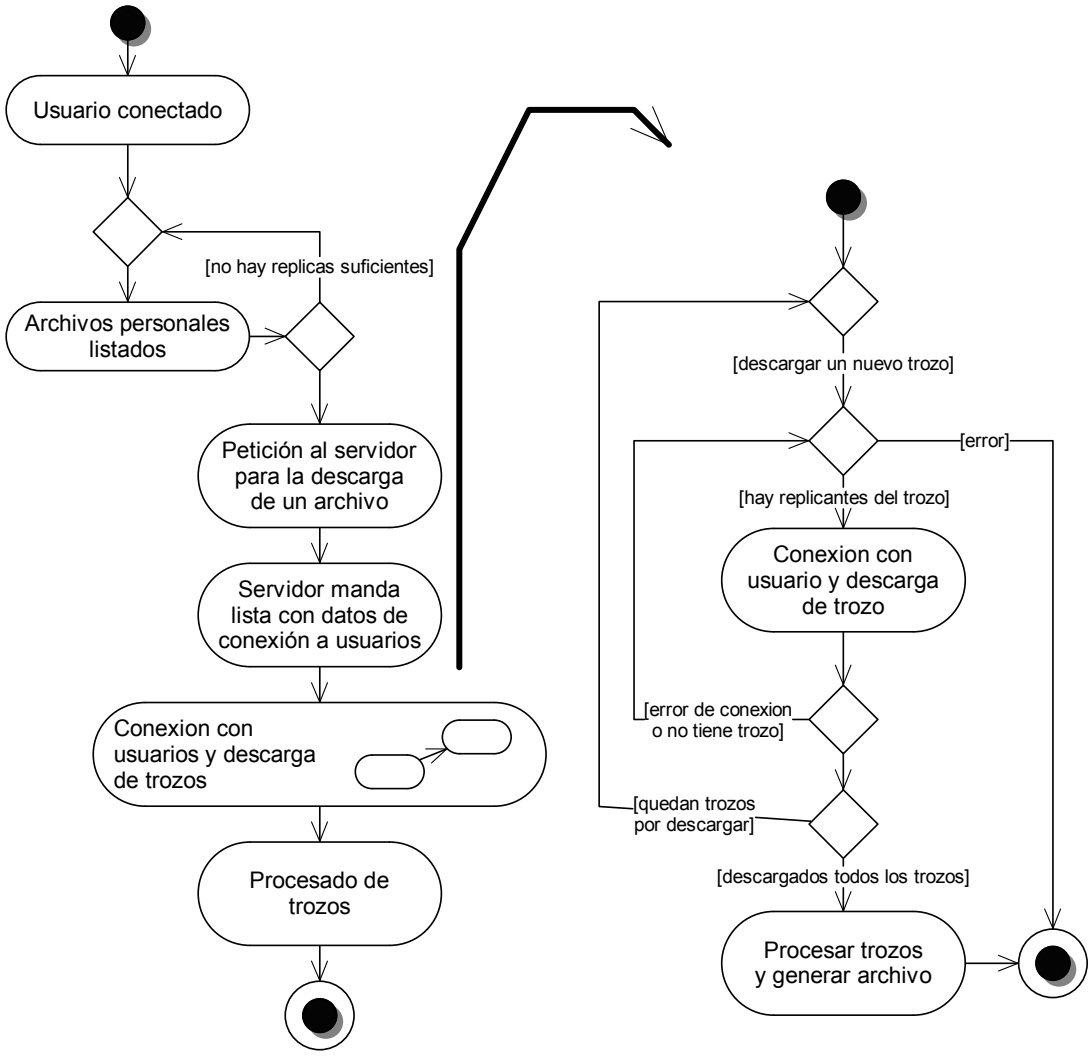


Figura 3.6.g: Diagrama de actividad de "recuperar archivos personales replicados"

El cliente se conecta a la red P2P y lista sus archivos personales. Con esto sabe cuántos replicantes de sus trozos hay en estos momentos conectados, y le da una "disponibilidad" de sus archivos.

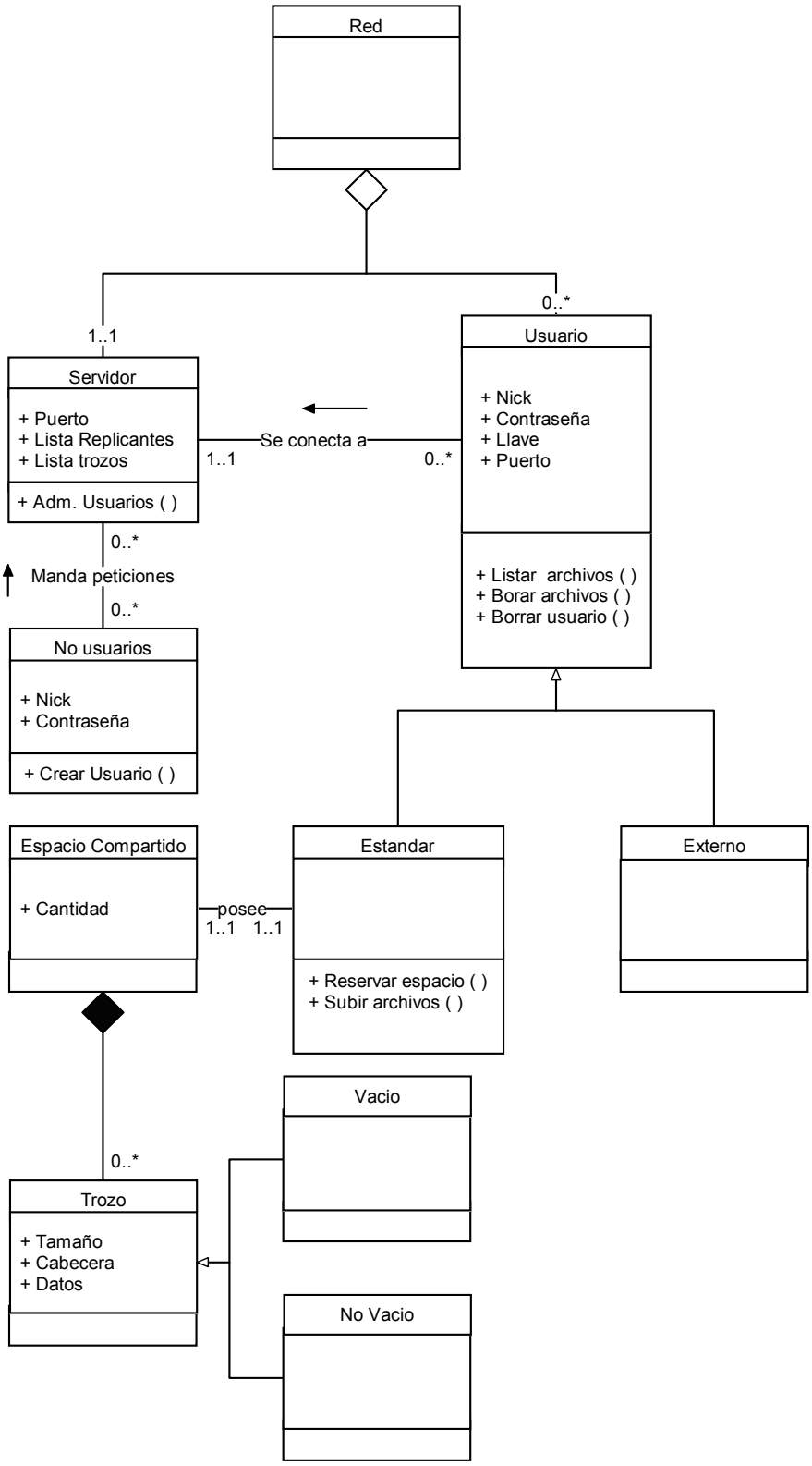
El usuario selecciona un archivo, y solo en el caso de que haya al menos un replicante de cada uno de los trozos le da la opción de iniciar la descarga.

El usuario le pide al servidor una lista de los replicantes de los trozos de ese archivo, y el servidor le contesta con los datos de conexión (Ips, puertos) de esos usuarios.

El usuario inicia la conexión y descarga de los trozos. Esto lo hemos dividido en un subproceso para profundizar más en cómo se desarrolla esta tarea. El usuario intenta conectar con el primer replicante del primer trozo y descarga este archivo; si ha tenido éxito continúa con el siguiente trozo, pero si no se pudo realizar la tarea correctamente, ya sea porque no pudo conectar con el usuario (que se hubiera desconectado justo en ese momento) o quizás no dispusiera del trozo que se suponía que sí tenia, pasa a intentar conectar con el segundo replicante de este trozo. En caso de que no hubieran más replicantes del mismo, significa esto que no podremos conseguir la replica de ese trozo en este momento, y por tanto no podremos conseguir la replica del archivo entero (ya que necesitamos todos sus fragmentos para poder rehacer el archivo) así que cancelamos la descarga.

Suponiendo que todo haya ido bien y tengamos todos los trozos que componen el archivo, usamos la llave para descifrarlo y recomponerlo, y ya tenemos el archivo completo.

3.7. DIAGRAMA DE CLASES



4. DISEÑO

4.1. DISEÑO DEL SISTEMA DE ARCHIVOS

4.1.1. INTRODUCCIÓN

Como ya hemos desarrollado en el capítulo anterior, existe un sistema de archivos específico para nuestra red P2P.

Un primer elemento es un "espacio compartido", que es una cantidad de almacenamiento físico que el usuario comparte para la red P2P. Para que este espacio compartido realmente sea usado por la red P2P, y no se consuma dicho espacio por la actividad de otros programas en uso en el ordenador, la idea es "ocupar" dicho espacio con datos "vacíos", que llamamos "trozos de archivo".

Por otro lado el sistema tiene la necesidad de fraccionar los archivos que se quieran mandar por muchas razones (comentadas anteriormente), pero entre las más importantes estarían el reducir el tamaño de los archivos que se envían de forma continua lo que requiere conexiones poco prologadas, una mejor distribución proporcional en la red de archivos, facilidad de replicación, seguridad al no poseer un usuario una replica entera de toda la información del archivo, etc.

Así pues, existe un sistema de archivos (trozos), tanto para reservar el espacio compartido del usuario, como para el envío de las replicas de los archivos de los mismos. Todos los trozos, tanto si contienen información como si no, tienen el mismo formato y tamaño, y al estar cifrada la información esto redunda en la seguridad del sistema. Estos trozos constan de 3 partes: cabecera, datos y hash.

4.1.2. DISEÑO DE LA CABECERA

La cabecera debe constar de los siguientes elementos, al menos:

- Nombre del archivo al que pertenece: Necesario para que el usuario que tenga la llave y pueda descifrarla sepa a cual pertenece.
- Número de trozo: También necesario, ya que al fraccionar el archivo el fragmentos, necesitamos saber dicho "trozo" que posición ocupa dentro de esa lista de fragmentos
- Número total de trozos: Igualmente necesario, para saber dónde termina la lista de trozos, y cuál es el último.

La longitud del nombre de archivo puede tener una extensión considerable, sobre todo en las últimas versiones de Windows, aunque rara vez sobrepasa los 256 caracteres, lo que por otra parte suele ser el tamaño máximo de nombre de archivo en muchos sistemas. Así que podemos fijar un tamaño de 256 caracteres máximo para el nombre de archivo, lo cual es bastante razonable, y si algún archivo sobrepasara dicho tamaño, truncar el nombre de archivo, conservando la extensión del mismo.

Por otro lado esta el número de trozo, y el número total de trozos del archivo. Para este fin se ha creado un sistema de codificación en base 256 (un byte) para representar cantidades numéricas, que luego se usará además en muchos otros puntos del sistema. Con un byte podemos contar hasta 256 trozos, con dos 256 * 256, … ¿Cuántos bytes usar? Eso dependerá del tamaño de los datos de cada trozo, y del tamaño de datos del sistema, pero en principio usaremos 4 bytes, que no es un tamaño excesivo, y con el que tenemos capacidad para representar números hasta 256 elevado a cuatro, es decir, números de hasta magnitud 4.294.967.296.

Aún así aun necesitamos otros elementos para la consistencia de la cabecera y los requerimientos del sistema:

- Tamaño de la cabecera: El tamaño algunos datos, como el nombre de archivo, no es fijo, por lo tanto la longitud de la cabecera no es fija. Necesitamos saber qué tamaño de datos útiles tiene nuestra cabecera.
- Usuario del archivo: El nombre de usuario del dueño del archivo es necesario para el correcto funcionamiento del sistema y para eliminar ciertas inconsistencias, como por ejemplo la que se daría si dos usuarios intentan

compartir un fichero idéntico. Con el nombre del usuario como parte de la cabecera, dos archivos iguales, codificados con la misma "llave de codificación", no darían igual si son codificados por usuarios distintos.

- Caracteres de separación de campos variables: Como hemos comentado hay campos que son variables, y por lo tanto necesitamos algún carácter que controle la longitud de estos campos, dónde acaba uno y empieza otro.
- Tamaño del último trozo: Tenemos que hacer todos los trozos del mismo tamaño, así que es posible que el último trozo no llene el tamaño de trozo previsto. Si fuera el último trozo por lo tanto se rellenaría de datos vacíos después de los útiles, así que necesitamos saber el tamaño de los datos del último trozo, para saber que datos son útiles y cuales no.

La siguiente tabla nos muestra la composición de los distintos elementos dentro de la cabecera que hemos ido comentando, en el orden que la compondrá:

Elemento	**Tamaño Máximo**	**¿Tamaño Fijo?**
Tamaño cabecera	4 caracteres	Sí
Nombre de archivo	256 caracteres	No
Carácter Separación	1 carácter	Sí
Número de trozo	4 caracteres	Sí
Carácter Separación	1 carácter	Sí
Número de trozo	4 caracteres	Sí
Carácter Separación	1 carácter	Sí
Tamaño último trozo	4 caracteres	Sí
Nombre de usuario	27 caracteres	No
Relleno	Máx. Tam. Cabecera – Tot. Cabecera	No

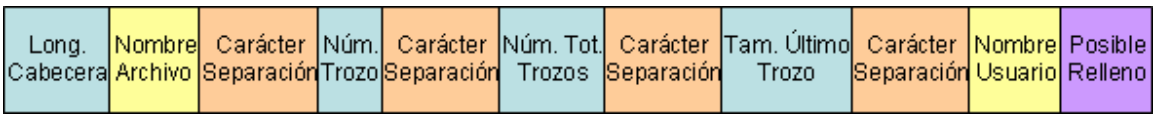


Figura 4.1.a: Formato de la cabecera de archivo.

Como vemos, la suma del tamaño máximo de los distintos componentes de la cabecera es de 302 caracteres, lo que será el tamaño fijo de las cabeceras de los trozos

de archivos. Necesitamos que el tamaño sea fijo ya que si no haría variable el tamaño de los trozos, así que si la cabecera poseyera un tamaño inferior a 302 caracteres se rellenaría con datos vacíos.

4.1.3. DISEÑO DEL HASH

A pesar de que el mecanismo de comunicación TCP ya incluye corrección de errores en la comunicación, y el propio sistema operativo Windows también posee su propio mecanismo contra la corrupción de errores, a veces estos errores se produce y no son detectados, ya sea de forma intencionada o no.

Por lo tanto, como un elemento añadido a la integridad del sistema, contra dicha posible corrupción de los datos se ha añadido un pequeño "hash" al final del archivo. Este "hash" ayuda a la detección de errores, complementándose con los ya existentes; pero la finalidad no ha sido implementar un complejo sistema de "hash" como los existentes hoy en día, capaces no solo de detectar errores sino incluso de corregirlos, como por ejemplo el I.C.H del Emule (Gestión Inteligente de Corrupción), sino más bien de agregar una nueva funcionalidad en el sistema que le de consistencia, y que en un futuro se pueda robustecer cuanto se requiera.

El "hash" de cada trozo se compone únicamente de un "byte" de información, que es el resultado de hacer un XOR de todos los bytes del trozo una vez codificado el mismo. Así, cuando un trozo se recibe de otro usuario, o cuando se quiere comprobar la integridad de los trozos almacenados es un proceso rápido y simple; basta con hacer un XOR de todos los bytes del "trozo", y si coincide el resultado con el hash que incorpora el archivo es que el archivo no está corrupto. La posibilidad de corrupción en si es ínfima debido a los mecanismos de seguridad del propio sistema operativo y del protocolo de comunicación, pero gracias a este "hash" existiría una posibilidad menor del 0,5 % de una corrupción no detectada (1/256), sobre esa posibilidad ínfima de corrupción.

A continuación pondré un ejemplo de funcionamiento del sistema hash:

Mensaje: "AVIÓN"
Mensaje codificado: "CASAS"
Este mensaje, pasado a bytes, y bits,

Carácter ASCII	***Byte***	***Bits***
C	*67*	*1000011*
A	*65*	*1000001*
S	*83*	*1010011*
A	*65*	*1000001*
S	*83*	*1010011*

Hacemos un XOR entre todos sus bytes, lo que es lo mismo entre todos sus bits carácter a carácter:

1000011 XOR 1000001 = 0000010
Resultado XOR siguiente byte: 0000010 XOR 1010011 = 1010001
1010001 XOR 1000001 = 0010000
0010000 XOR 1010011 = 1000011

El resultado sería 1000011, lo que corresponde con el carácter ASCII 67, que casualmente es un carácter fácilmente representable: "C". Por lo tanto el mensaje final sería "CASASC". El usuario no sabe que contiene el archivo, pero si puede comprobar su integridad siguiente que hemos seguido nosotros mismos para componer el hash. Si el resultado fuera otro distinto de "67", es que el mensaje está corrupto.

4.1.4. DISEÑO DE LOS DATOS DEL TROZO

Otro tema a tratar es el diseño de los datos contenidos en los trozos de archivo. Estos datos son el espacio que sobra quitando el tamaño de la cabecera y del hash, respecto del tamaño total que tengan los trozos de archivo.

¿Qué tamaño usar? El tamaño a usar en los "trozos" de archivo está relacionado con otro elemento del sistema que es el "tamaño" que el usuario quiere compartir de disco para la red P2P. Dicho tamaño se le pide al usuario en MB, que es la medida mas lógica considerando que si la diéramos en GB, un GB es mínimo muy elevada, y si la diéramos en KB, es quizás una medida muy escasa en relación al tamaño que tendrían los trozos.

Por otro lado el tamaño de los trozos, como comentamos anteriormente, tiene que ser un tamaño que permita enviarlos de forma rápida entre usuarios (que no obligue a mantener conexiones muy prolongadas), y al mismo tiempo que sean suficientemente grandes como para que el fraccionado de los archivos, y por tanto la inclusión de las cabeceras suponga un tamaño insignificante respecto a los datos útiles. Otro dato que hay que tener en cuenta es que, como queremos que los trozos tenga un tamaño fijo, si tuviéramos trozos de archivo muy grandes, debido a este "tamaño fijo", un fichero pequeño generaría un trozo mucho mas grande que él, siendo claramente ineficiente el sistema si se quisieran replicar muchos archivos de pequeño tamaño, como por ejemplo documentos de texto. Quedando por tanto descartado trozos de archivo mayores de un MB, y considerando que el formato en el que el usuario elige la cantidad de espacio compartido es en Megas, el tamaño de trozo deberá ser una fracción entera de un mega; la proporción de datos añadidos respecto de los datos útiles debe ser mínima (menor del 1%), y a la vez no demasiado grande para cumplir con las condiciones anteriormente mencionadas.

Por todo esto, finalmente se opta por un fichero de tamaño 128KB, lo que es la octava parte de un mega, y cumple todos los requerimientos impuesto al formato de "trozos". 128KB, son 128 * 1024 bytes, lo que hace un tamaño de trozo de 131072 bytes, lo que restando el tamaño de la cabecera, 302 bytes, y el tamaño del hash, 1 byte, hace un tamaño útil de datos de 130769 bytes, y los datos añadidos (cabecera y hash) representan aproximadamente el 0,23% de los datos del trozo.

4.2. DISEÑO DEL INTERFAZ DE PROGRAMA

4.2.1. DISEÑO DEL INTERFAZ DEL CLIENTE

El aspecto del interfaz del cliente es el siguiente:

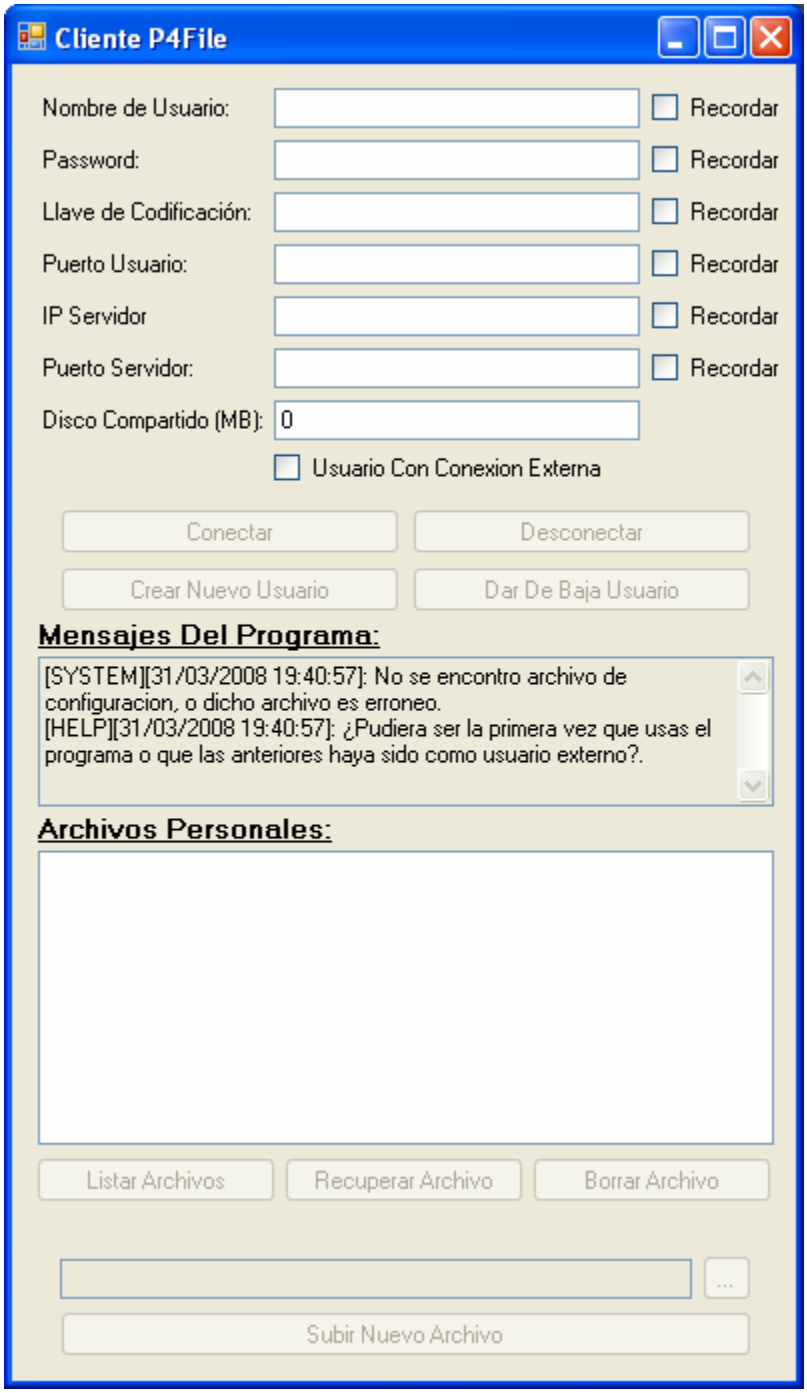


Figura 4.2.a: Captura de la intefaz gráfica del cliente.

Este interfaz satisface todas las necesidades del cliente de la red P2P. Tiene cajas de texto para introducir los datos requeridos por el sistema:

- Nombre de usuario con el que se conecta a la red.
- Password o contraseña con la que se identifica ante el servidor.
- Llave de codificación con la que están cifrados sus archivos personales, y con la que se descifrarán
- Puerto de usuario, el cual usará para escuchar las peticiones de otros clientes. Es obligación del cliente tener habilitado dicho puerto para conexiones entrantes P2P a través de firewalls u otros elementos ajenos al software.
- Dirección IP del servidor al que se quiere conectar.
- Puerto del servidor al que se quiere conectar.
- Tamaño de disco que comparte con la red P2P.

Estos son los elementos principales del usuario. Como se puede observar a la derecha de cada campo hay unos *checkbox* (casillas para marcar), los cuales si son marcados, al cerrar el programa, se recordarán para que en futuras ejecuciones del mismo no sea necesario introducir esos datos y aparezcan ya rellenos de forma automática. También se puede observar que el único campo que no tiene checkbox es el espacio compartido, ya que este, si hay, es recordado de forma automática.

También se puede observar justo debajo de todos estos datos, un checkbox que se llama “usuario con conexión externa”. La funcionalidad de este checkbox es exactamente la que su nombre indica: cuando un usuario, que ya tiene una cuenta creada en el servidor se quiere conectar pero no está en su ordenador personal no dispone ahí de su “espacio compartido”. Si se conectara de forma normal, el servidor interpretaría que el usuario por la razón que sea ha borrado o perdido todos los “trozos” de su espacio compartido, y por tanto todas las replicas de archivos personales que allí hubiera. Gracias a esta opción un usuario se puede conectar desde terminales distintas a la suya habitual y acceder a la red para descargar sus archivos. No podrá realizar algunas de las funcionalidades que tienen los usuarios “estandar”, como “ampliar su espacio compartido”, o poder mandar fragmentos solicitados por otros usuarios (ya que no dispone de los archivos requeridos), pero podrá usar aquellas funciones que no requieren de ese espacio compartido como por ejemplo descargar sus propios archivos personales.

Las cajas de texto para introducir los datos, tienen algunos otros elementos interesantes, como por ejemplo que en los datos de "puerto" solo te deja introducir números, y nunca mas de 5 caracteres, en el dato de IP del servidor no permite introducir mas de 15 caracteres (correspondiendo al máximo de 4 números de 3 cifras y 3 puntos), etc.

Un poco más abajo tenemos los botones "conectar", "crear usuario", "desconectar" y "dar de baja usuario", correspondientes a las funciones de similar nombre del sistema. Como podemos apreciar en la figura anterior, en ese momento concreto de la captura estaban desactivados, ya que otras de las características del interfaz es que interactúa con el estado del cliente y automáticamente va cambiando el estado de interfaz.. Por ejemplo, los botones "conectar" y "crear nuevo cliente" no se activan hasta que no están rellenos todos los datos del cliente; sin embargo, si activamos la casilla de "conectar como usuario externo", no nos hace falta el dato del "puerto de usuario" (ya que no vamos a escuchar peticiones de otros clientes) y por tanto el interfaz sí activa los botones de "crear usuario" y "conectar" aunque no se encuentre introducido dicho dato. Otro ejemplo: el sistema no permite "llaves" de cifrado menor de 8 cifras como medida de seguridad, así que si la elegida tiene menos de 8 cifras, no permite la conexión del cliente. También podemos observar como los botones "desconectar" y "dar de baja usuario" no se activan hasta que el usuario no se haya conectado con el sistema, de la misma forma que "conectar" y "crear usuario" se desactivan cuando esto sucede. En las siguientes figuras se muestran algunos ejemplos lo comentado.

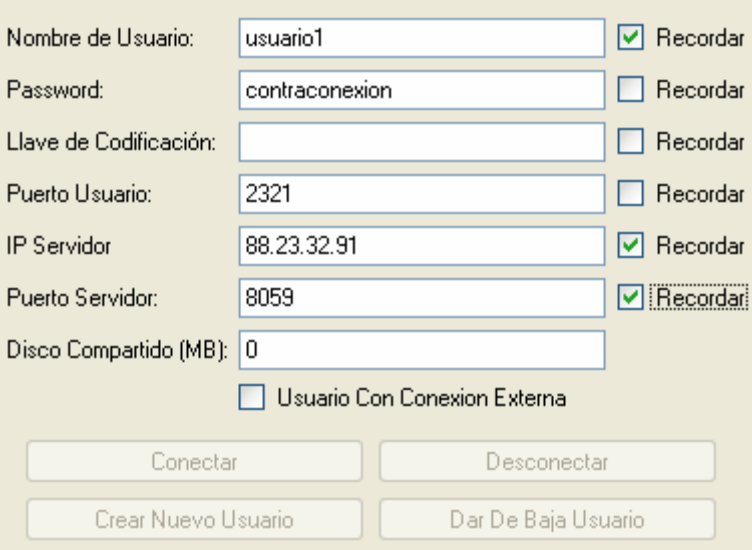


Figura 4.2.b: Faltan datos: botones desactivados

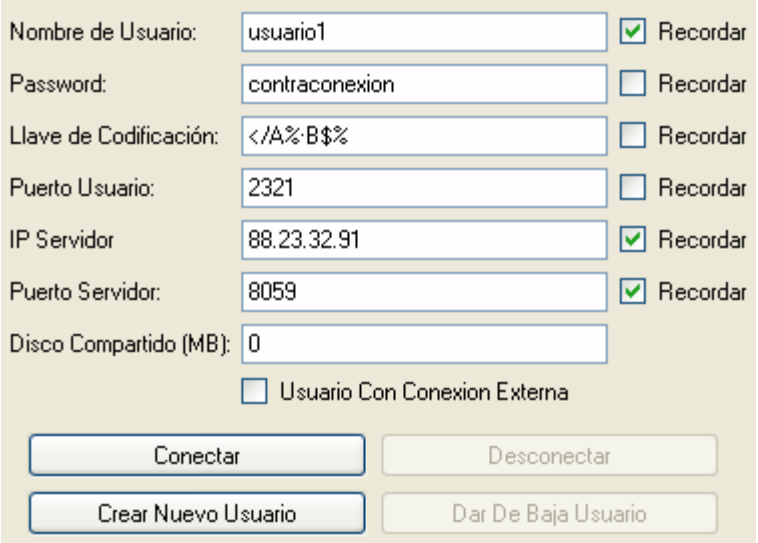


Figura 4.2.c: Datos necesarios introducidos: se activan los botones

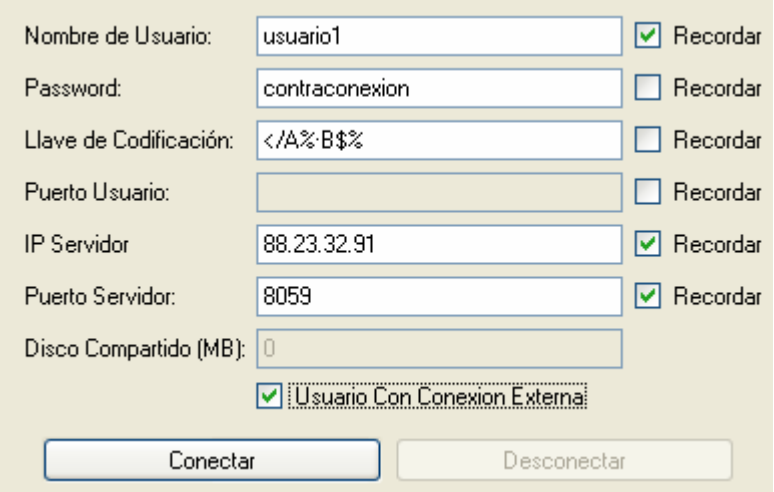


Figura 4.2.d: Usuario con conexión externa.
Se activa conectar aunque falta el "puerto".

El siguiente elemento que nos encontramos en el interfaz de cliente es una ventana de texto con mensajes del programa. Es un elemento muy interesante para saber en qué estado nos encontramos en cada momento (a pesar de que el sistema ya genera un "log" de errores). En esta ventana de texto aparecen dos tipos de mensajes: mensajes de sistema, donde este nos dicen los diferentes acontecimientos producidos en el mismo (conexiones, desconexiones, ficheros descargados, etc), y mensajes de ayuda que vienen acompañados en ciertos mensajes de sistema, y que nos indican las posibles causas de ciertas situaciones. Por ejemplo, en la figura donde se ha mostrado

el aspecto del interfaz del cliente, se puede observar dos mensajes, uno de sistema (con fecha y hora incluida) donde el sistema nos dice que no encuentra el fichero donde se guardan los datos del usuario (lógico, ya que es la primera vez que se ejecuta el cliente), y en un segundo mensaje de ayuda, nos dice dos de las posibles causas por las que esto pueda pasar: porque es la primera vez que se ejecuta el cliente, o quizás porque las anteriores veces se ejecuto como usuario externo y no se guardaron los datos.

Los siguiente elementos que nos encontramos son relacionados con los "archivos personales del usuario". Tenemos un "listbox" (una caja donde se muestran los elementos de una colección) donde, si hacemos una petición de listar nuestros archivos personales, se muestran éstos y su disponibilidad. Más abajo tenemos tres botones "listar", "recuperar", "borrar", todos ellos referentes a nuestros archivos personales. "Listar" ejecuta una petición que se manda al servidor para listar nuestros archivos personales (en la figura se encuentra desactivado, porque el cliente no está conectado al servidor). "Recuperar" solo se activa si tenemos seleccionado un archivo en el "listbox" de archivos que tenga mínimo disponibilidad "1", es decir, mínimo una replica disponible de cada uno de los "trozos" del archivo, lo que nos permitiría recuperar esos trozos y reconstruir el archivo); una vez pulsado inicial el proceso de recuperar todos los fragmentos de ese archivo personal. "Borrar", como su nombre indica, ejecuta la petición de borrar el archivo personal seleccionado.

Y por último tenemos tres elementos relacionados con subir nuevos archivos personales a la red P2P, una barra de texto, y dos botones. El botón llamado "…" abre un explorador de Windows para que podamos buscar el archivo que queremos subir a la red; una vez seleccionado añade la dirección completa en la caja de texto, en cual también podría haberse escrito a mano dicha dirección. Solamente si el archivo existe se activa el botón "subir nuevo archivo", que nos proporciona la funcionalidad ya descrita anteriormente.

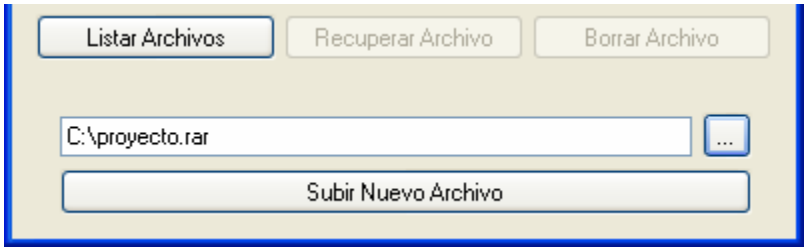


Figura 4.2.e: Archivo seleccionado. Botón subir activado

4.2.2. DISEÑO DEL INTERFAZ DEL SERVIDOR

El servidor realmente no precisa de un interfaz concreto para ejecutar las tareas encomendadas, pudiera haberse diseñado como un programa residente en memoria. Sin embargo se ha optado por un diseño con un sencillo interfaz para tener mejor control sobre los sucesos en la red.

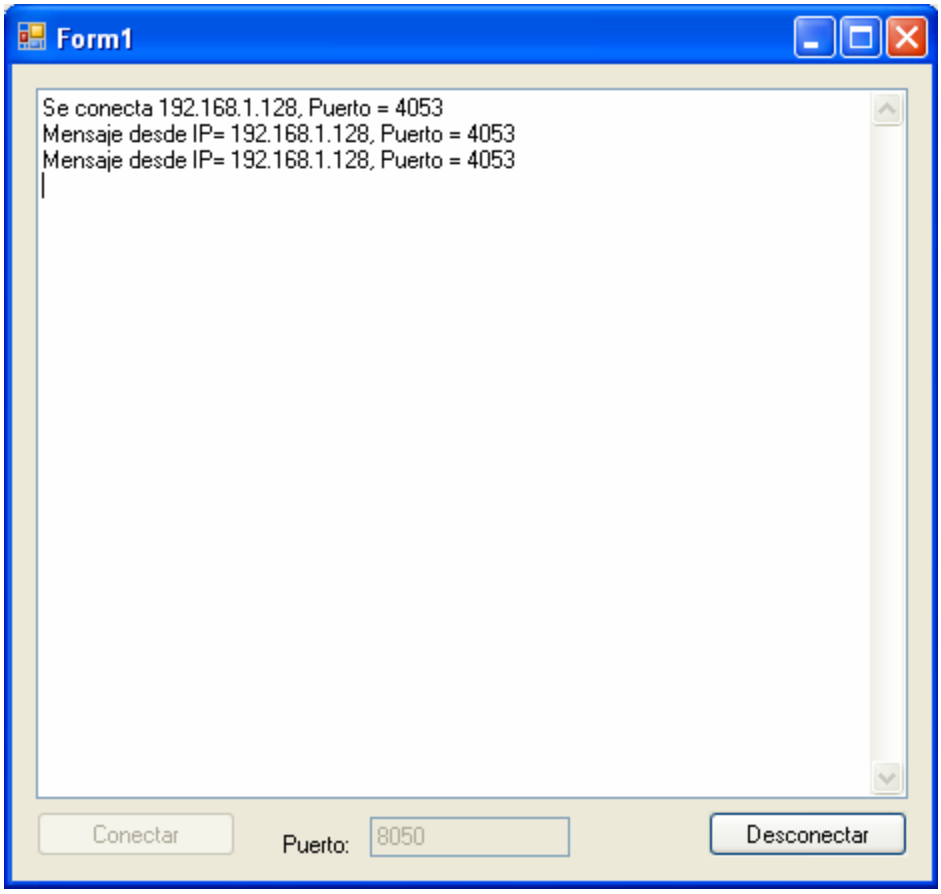


Figura 4.2.f: Servidor conectado.

Como se puede observar en la captura de pantalla, el interfaz del servidor se compone de unos pocos elementos. El primero es una ventana de texto que muestra mensajes referentes a qué clientes se conectan o se desconectan, o si mandan algún mensaje. Debajo de dicha ventana tenemos un botón de conectar, que activa la función de escuchar peticiones de conexión de nuevos clientes, a través del puerto indicado en la caja de texto de la derecha, la cual solo permite introducir valores numéricos. Y por último un botón de "desconectar", que cierra las conexiones con todos los clientes que estuvieran conectados.

4.3. DISEÑO DE LOS DATOS

4.3.1. DISEÑO DE LOS DATOS MANEJADOS POR EL CLIENTE

Tanto servidor como cliente manejan ciertas estructuras de datos para múltiples tareas, los cuales almacenan en archivos binarios para su uso.

En el caso del cliente hay dos tipos de ficheros de datos que maneja: un fichero de configuración de usuario, y dos ficheros de "log" de la aplicación.

Sobre el primero de ellos ya hemos hablando en otras secciones de la memoria (en el diseño de la interfaz por ejemplo). Al cliente se le ofrece la posibilidad de "recordar" aquellos datos de usuario que él mismo decida, marcando la casilla adyacente a dicho dato en el interfaz del cliente, con lo que la próxima vez que ejecute el cliente, esos datos se rellenarán de forma automática, suponiendo esto un valor añadido para el cliente. Dichos datos a la hora de cerrar el cliente se almacenan en un fichero llamado por defecto "user.cnf", siempre que el usuario no tenga marcado la casilla "usuario externo" en cuyo caso se presupone que no está en su ordenador habitual y no desea que se guarden esos datos.

Por otro lado, el cliente guarda en el sistema un "log" (si no está conectado como usuario externo) de todos aquellos eventos ocurridos durante la ejecución del cliente. Este "log" no es ni más ni menos que los mensajes mostrados por el sistema en la ventana que está dispuesta para tal fin en el interfaz de cliente, y que se almacena en un archivo de texto llamado "log.txt", anexándose a contenidos previos si los hubiera. Por último también guarda un "log" más detallado con descripción técnica de todas las excepciones de sistema. Estos "log" pueden ser de gran utilidad para el cliente ya que indica la fecha y hora de todo lo acontecido, pudiéndose revisar posteriormente para su estudio.

4.3.2. DISEÑO DE LOS DATOS MANEJADOS POR EL SERVIDOR

Por la filosofía centralizada de la red P2P, el servidor maneja una considerable cantidad de datos listas y ficheros, sobre todo respecto al cliente. Estos ficheros de datos son tres: un fichero con los datos de los usuarios, un fichero con los trozos "no vacíos" de los usuarios, y otro con una lista de replicantes de dichos trozos.

El primero de los ficheros que maneja el servidor es un fichero con los datos principales de los usuarios de la red. En este fichero se almacena el nombre de usuario, la contraseña de conexión, y la cantidad de espacio que dicho usuario comparte a la red. Dicho archivo es consultado y/o editado cuando un usuario se autentifica en la red, cuando quiere cambiar su cantidad de espacio compartido o cuando se quiere dar de baja dicho usuario. Tiene por defecto el nombre de "users.dtf" ("users data file").

El segundo archivo es una lista de los "trozos no vacíos" que tiene el usuario en su espacio compartido. Como ya hemos comentado con anterioridad, una de las medidas de seguridad de la red consiste en que el usuario no sabe cuáles de los ficheros que están en su espacio compartido, y que tienen todos el mismo tamaño, son ficheros con datos útiles, y cuáles son ficheros huecos o vacíos. Por esta razón, es el servidor quien tiene que encargarse de dicha tarea, y de que cuando se conecte el cliente al servidor, compruebe si dicha lista se mantiene o si, por las causas que sean alguno de esos archivos ya no se encuentra presente en el espacio compartido del usuario, con lo que el usuario ya no es "replicante" de ese trozo.

El servidor necesita identificar de forma unívoca cada trozo para poder asociarlo con su dueño, pero al mismo tiempo se pretende que el servidor no tenga conocimiento de lo que dicho archivo contiene. Y precisamente las cabeceras usadas para la codificación y fragmentación de los archivos cumplen ese fin, ya que llevan datos que lo distinguen incluso de otro usuario que compartiera el mismo archivo pero están cifradas para que solo el legítimo dueño sepan que contiene.

Por lo tanto el servidor almacena en el archivo "comp.dtf" las cabeceras de trozos "no vacíos" que el usuario tiene almacenadas, aunque no sabe el contenido de dichas cabeceras ya que están cifradas y el cliente en ningún momento le comunica dicha clave de cifrado. Dichos archivos "no vacíos" llegan al cliente cuando otros clientes suben a su espacio compartido fragmentos de las replicas de sus ficheros,

pero no es el cliente receptor quien comunica al servidor que tiene ese nuevo archivo, si no el propio usuario que se lo ha mandado, con lo que este usuario receptor no sabe cuáles de sus archivos son vacíos o no, y se mantiene la integridad de este elemento de seguridad del sistema.

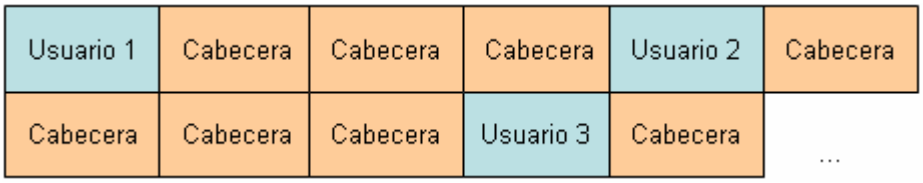


Figura 4.3.a: Ejemplo de la estructura del archivo de "trozos no vacíos"

Por último, el servidor maneja un tercer fichero que, si bien no sería estrictamente necesario en el sistema (aunque para ello el archivo "comp.dtf" debería estar organizado de forma distinta), sí agiliza enormemente el funcionamiento del mismo; es una lista de replicantes. La lista de replicantes es realmente una lista inversa de la lista de "trozos no vacíos de los usuarios", ya que su estructura, en vez de ser "usuario" => "trozos no vacíos del usuario", es "usuario dueño de los trozos" => "trozos" => "usuario que replica dichos trozos". Esta lista es fundamental para el ágil funcionamiento del servidor ya que, por ejemplo, cuando un usuario quiere listar sus archivos, y le pide al servidor que le mande la lista de cabeceras que lo componen, el servidor no tiene que ir analizando usuario por usuario, mirando su lista de trozos replicados, y comprobando si alguno pertenece al usuario que realiza toda la petición (con lo que debería recorrerse la lista entera), sino que le basta con localizar al usuario en esta nueva lista, y ahí tiene toda la lista de trozos de forma secuencial.

Figura 4.3.b: Ejemplo del archivo de replicantes

4.4. DISEÑO DE LA SEGURIDAD

Durante todo el documento se ha ido desgranando los diferentes elementos relacionados con la seguridad del sistema. Como ya hemos dicho anteriormente, la finalidad del proyecto no es crear un robusto sistema de seguridad de anonimato infranqueable, pero si aprovechar la implementación del mismo para incluir en la red de archivos distribuidos mediante P2P elementos de seguridad que ayuden en la medida de lo posible a mejorar el funcionamiento y la privacidad.

Estos elementos de seguridad principales son los siguientes:

- Cifrado de archivos y cabeceras.
- Fraccionado de archivos.
- Clave de conexión, y llave de cifrado de archivos.
- Espacio compartido.
- Hash.

Uno de los elementos indispensables para este sistema P2P es un cifrado para los archivos debido a que estos se hayan almacenados en ordenadores ajenos y podrían ser objeto de miradas indiscretas. Tanto los archivos (los trozos de archivo) como las cabeceras que los describen son cifradas mediante un algoritmo de cifrado de tipo "cifrador de flujo de bits". Es decir, se cifra todo el contenido mediante una llave de cifrado bit a bit con una función XOR. Es un cifrado sencillo y rápido, pero a la vez potente a menos de que se disponga de la llave de cifrado. Podría haberse empleado cualquier otro algoritmo de cifrado más robusto (.net dispone de una amplia variedad de funciones criptográficas), como DES, AES, RSA… pero la pretensión del proyecto, como se comentó anteriormente, no es crear un sistema de cifrado "infranqueable", si no tocar todas las ramas presentes en un software P2P, en este caso la seguridad de la información. Este cifrado es suficientemente fuerte para cumplir el papel de "cifrado de la información del proyecto", y si en un futuro se quiere implementar solo hay que cambiar la función de que devuelve el resultado cifrado. Además, la llave de cifrado solo la posee el usuario (y es distinta, a menos de que usuario quiera lo contrario, de la clave de conexión), nunca el servidor ni otros usuarios con lo que con lo que en principio el sistema ese seguro, salvo ataques malintencionados.

Otro de los elementos que influye en la seguridad del sistema es el fraccionado de archivos. Uno de los problemas por el tipo de sistema P2P implementado es que otros usuario tienen las replicas de los archivos personales del cliente, de ahí la necesidad del cifrado de los mismos que hemos comentado antes. Pero como los archivos son fraccionados, y el replicado de esas fracciones se realiza en distintos usuarios, nunca un usuario (a menos de que la red tuviera solo uno) posee todos los fragmentos para recomponer el archivo (y aunque los tuviera, estaría cifrado igualmente) con lo que aun el caso de descifrarlos solo tendría fragmentos inconexos de información. Este es otro de los puntos que aumenta la seguridad del sistema.

Por otro lado, como ya se ha explicado en otras secciones, el espacio compartido se rellena de archivos huecos, que no casualmente tienen el mismo tamaño y forma que los archivos “no huecos”, así que cuando estos reemplazan a los archivos huecos, todos tienen la misma forma. De hecho, los archivos huecos incluso van cifrados con una clave generada aleatoriamente, así que un estudio malintencionado sobre la estructura de los archivos contenidos en ese directorio tendría más difícil averiguar cuáles son archivos cifrados con información, y cuáles son archivos huecos, lo que es un punto a favor de la seguridad del sistema.

Por último tenemos un *hash* del que ya se ha hablado en la sección “diseño del hash”. Este es otro de los elementos que aportan consistencia al sistema, ya que detecta la corrupción de los datos, malintencionada o no, tanto durante la comunicación como posteriormente mientras se encuentran los datos almacenados en el cliente.

5. GESTIÓN DEL PROYECTO

5.1. PLANIFICACIÓN DEL PROYECTO

5.1.1. RECURSOS Y TAREAS. HOJA DE RECUROSOS. DIAGRAMA WBS

Para la planificación del proyecto necesitamos definir primero el alcance de las tareas y recursos de los que se dispone.

Los recursos de los que se dispone para la realización del proyecto son los siguientes:

- Recursos humanos:
  - Un Jefe de Proyectos
  - Un Analista
  - Un programador especialista
  - Dos programadores
  - Un especialista en pruebas

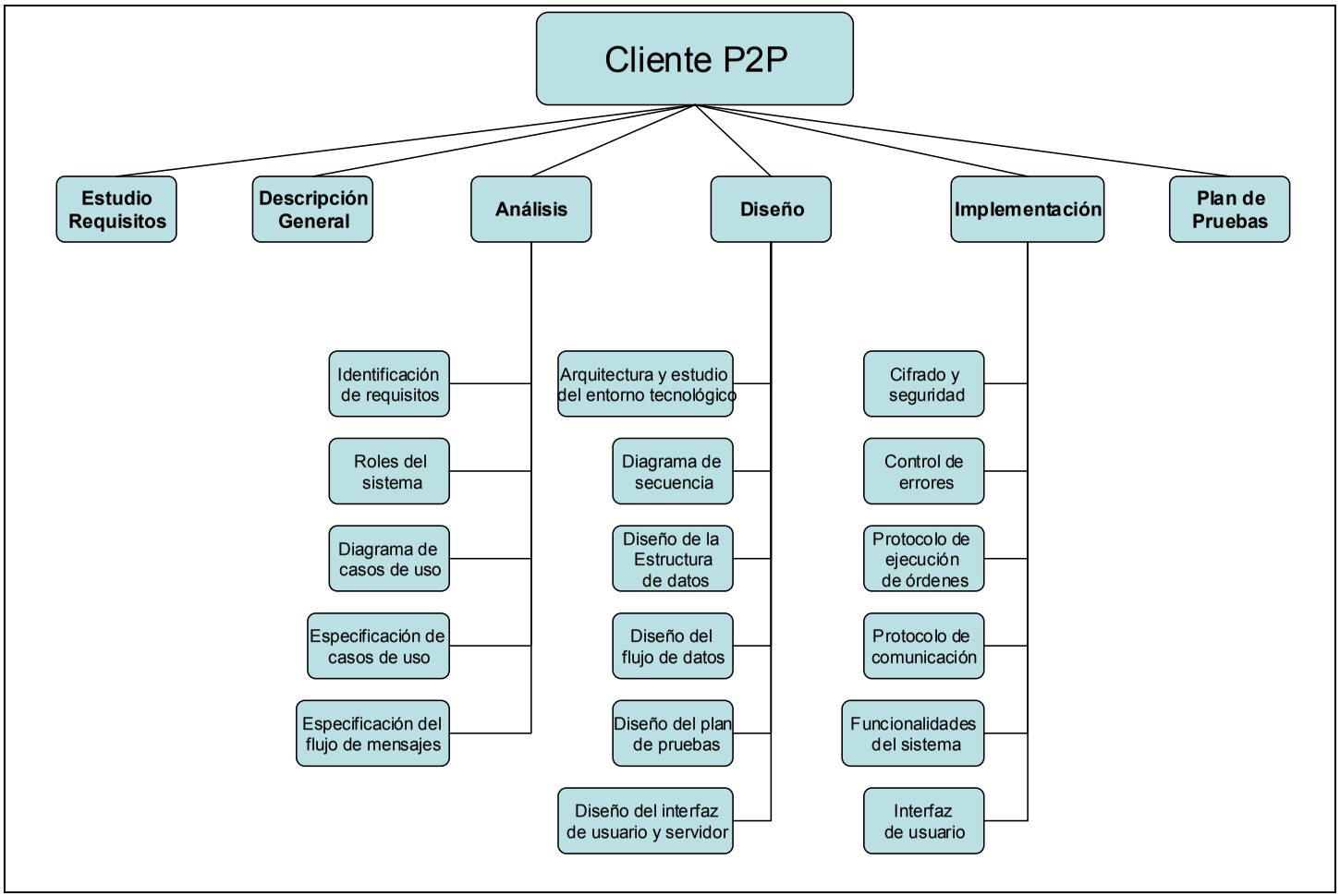
- Recursos materiales:
  - Tres ordenadores
  - Un servidor
  - Elementos de red para la comunicación.

A continuación se ofrece la hoja de recursos, con sus costes asociados, así como un diagrama WBS donde se organiza el trabajo requerido para la ejecución del proyecto:

HOJA DE RECURSOS

Nombre del recurso	Tipo	Etiqueta de material	Iniciales	Grupo	Capacidad máxima	Tasa estándar	Tasa horas extra	Costo/Uso	Acumular	Calendario base
Jefe de proyectos	Trabajo		J		100%	35,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Analista	Trabajo		A		100%	20,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Programador 1	Trabajo		P		100%	15,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Programador 2	Trabajo		P		100%	10,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Programador 3	Trabajo		P		100%	10,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Especialista en pruebas	Trabajo		E		100%	9,00 €/hora	0,00 €/hora	0,00 €	Prorrateo	Estándar
Ordenadores	Material		O			1.800,00 €		0,00 €	Prorrateo	
Servidor	Material		S			800,00 €		0,00 €	Prorrateo	
Red y Conexiones	Material		R			400,00 €		0,00 €	Prorrateo	

DIAGRAMA WBS



El diagrama WBS muestra las tareas identificadas para completar el proyecto. Muchas de dichas tareas ya han sido descritas durante el documento, así que pasamos a hacer un breve resumen de ellas:

- Estudio de requisitos: Se corresponde con el "estado de la cuestión", el estudio de las tecnologías existentes relativas al software que se pretende implementar, etc.
- Descripción general: Se define el concepto de software que se quiere hacer, funciones globales, cómo se quiere que funcione, etc.
- Análisis: En este apartado identificamos los requisitos del Cliente P2P, las funcionalidades que tendrá el mismo de manera global, los agentes involucrados en el sistema y cómo interactúan con el… Representamos las funcionalidades con varios tipos de diagramas, como los diagramas de secuencia o diagramas de clases, que definen cómo se deberían comportar las distintas funcionalidades del sistema. Con la especificación de los casos de uso profundizamos en la secuencia de pasos que realiza el sistema en sus distintas funcionalidades. Y por último definimos el flujo de mensajes relativo al funcionamiento del software, necesario al ser un software distribuido.
- Diseño: Primero nos involucramos en el aprendizaje del nuevo entorno de programación, en cuál hasta ahora es desconocido por los programadores; se necesita un tiempo considerable de aprendizaje, ya que se tocan elementos diversos del lenguaje, como "hilos", "sockets", cifrado, etc, además de los elementos clásicos de todos los lenguajes. Se profundiza en las distintas funcionalidades del software, en cómo este se comunica, cómo almacena los datos, que tipos de elementos requiere el interfaz para dar respuesta a las necesidades analizadas... También se diseña un plan de pruebas para detectar errores durante el desarrollo y testeo de la aplicación de la forma más eficiente posible.
- Implementación: Se implementa el software, desde los módulos más independientes como el cifrado, el protocolo de comunicación, etc, que se integran en el desarrollo de las distintas funcionalidades del sistema.
- Plan de pruebas: La última tarea definida, que será paralela a toda la implementación, es un plan de pruebas completo relativo a todas las funcionalidades desarrolladas.

5.1.2. DIAGRAMA GANT.

Una vez identificadas las tareas y los recursos disponibles para desarrollar el proyecto software, procedemos a organizar estas tareas temporalmente en base a los recursos y sus interdependencias temporales. Para esta tarea usamos un “Diagrama de Gantt”.

Para desarrollar este diagrama de Gantt hemos usado “Microsoft Project”, que permite crear dependencias temporales entre las tareas, asignarles una duración, qué recursos participan en la consecución de la misma, y en base a estos datos crear un “Diagrama de Gantt” que nos muestra la ejecución ideal del proyecto en base al tiempo.

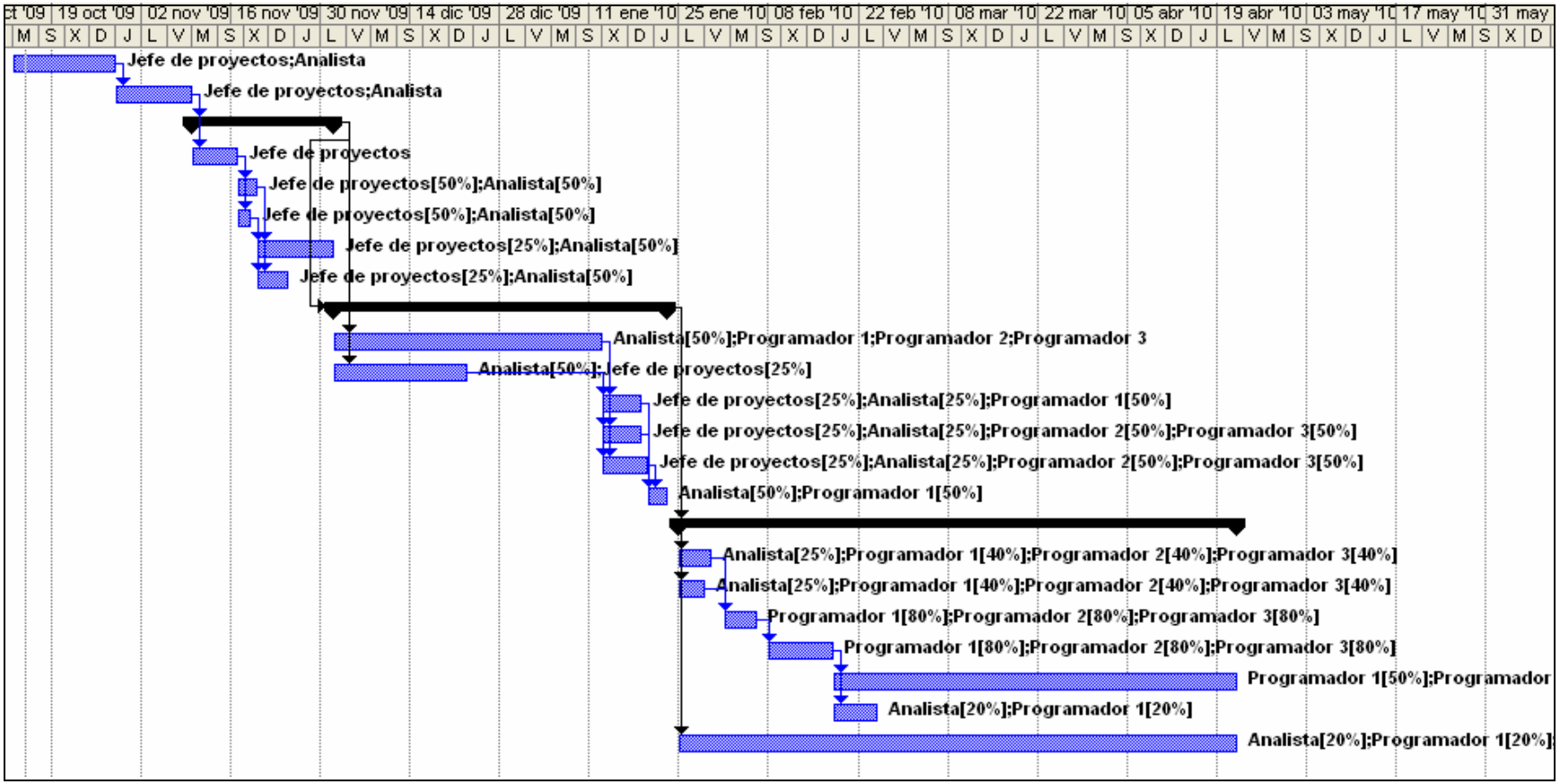
A continuación mostramos dos gráficas creas con “Microsoft Project”. La primera es la descripción de las tareas identificadas en el WBS, el tiempo que lleva su ejecución (son días laborables, con lo que 10 días serían dos semanas, por ejemplo), sus interdependencias y qué recursos participan.

La segunda figura es la representación gráfica de estos datos, lo que se conoce como digrama de Gantt. Podemos observar en el de manera visual más fácilmente las dependencias entre las tareas, aquellas que se pueden ejecutar de forma paralela, su ubicación temporal, etc.

DATOS PARA DIAGRAMA DE GANTT

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Estudio de requisitos	12 días	mar 13/10/09	mié 28/10/09		Jefe de proyectos;Analista
Descripción general	8 días	jue 29/10/09	lun 09/11/09	1	Jefe de proyectos;Analista
Análisis	**16 días**	**mar 10/11/09**	**mar 01/12/09**	**2**	
Identificación de requisitos	5 días	mar 10/11/09	lun 16/11/09	2	Jefe de proyectos
Roles del sistema	3 días	mar 17/11/09	jue 19/11/09	4	Jefe de proyectos[50%];Analista[50%]
Diagrama de casos de uso	2 días	mar 17/11/09	mié 18/11/09	4	Jefe de proyectos[50%];Analista[50%]
Especificación de casos de uso	8 días	vie 20/11/09	mar 01/12/09	5;6	Jefe de proyectos[25%];Analista[50%]
Especificación del flujo de mensajes	3 días	vie 20/11/09	mar 24/11/09	5;6	Jefe de proyectos[25%];Analista[50%]
Diseño	**38 días**	**mié 02/12/09**	**vie 22/01/10**	**3**	
Arquitectura y estudio del entorno tecnológico	30 días	mié 02/12/09	mar 12/01/10	3	Analista[50%];Programador 1;Programador 2;Programador 3
Diagramas de secuencia	15 días	mié 02/12/09	mar 22/12/09	3	Analista[50%];Jefe de proyectos[25%]
Diseño de la estructura de datos	4 días	mié 13/01/10	lun 18/01/10	10;11	Jefe de proyectos[25%];Analista[25%];Programador 1[50%]
Diseño del flujo de datos	4 días	mié 13/01/10	lun 18/01/10	10;11	Jefe de proyectos[25%];Analista[25%];Programador 2[50%];Programador 3[50%]
Diseño de plan de pruebas	5 días	mié 13/01/10	mar 19/01/10	10;11	Jefe de proyectos[25%];Analista[25%];Programador 2[50%];Programador 3[50%]
Diseño del interfaz de usuario y servidor	3 días	mié 20/01/10	vie 22/01/10	12;13;14	Analista[50%];Programador 1[50%]
Implementación	**63 días**	**lun 25/01/10**	**mié 21/04/10**	**9**	
Mecanismos de crifrado y seguridad	5 días	lun 25/01/10	vie 29/01/10	9	Analista[25%];Programador 1[40%];Programador 2[40%];Programador 3[40%]
Control de errores	4 días	lun 25/01/10	jue 28/01/10	9	Analista[25%];Programador 1[40%];Programador 2[40%];Programador 3[40%]
Protocolo de ejecución de órdenes	5 días	lun 01/02/10	vie 05/02/10	17;18	Programador 1[80%];Programador 2[80%];Programador 3[80%]
Protocolo de comunicación	8 días	lun 08/02/10	mié 17/02/10	19	Programador 1[80%];Programador 2[80%];Programador 3[80%]
Funcionalidades del sistema	45 días	jue 18/02/10	mié 21/04/10	20	Programador 1[50%];Programador 2[80%];Programador 3[80%]
Implementación de interfaz de usuario	5 días	jue 18/02/10	mié 24/02/10	20	Analista[20%];Programador 1[20%]
Plan de pruebas globales	63 días	lun 25/01/10	mié 21/04/10	9	Analista[20%];Programador 1[20%];Programador 2[20%];Programador 3[20%];Especialista

DIAGRAMA DE GANTT



5.1.3. PRESUPUESTO DEL PROYECTO

Definidos los recursos de los que disponemos para la conclusión del proyecto, así como las tareas y su duración, procedemos a realizar un cálculo estimativo del coste del proyecto si no hay desviaciones de costes, y suponiendo que el cálculo temporal de finalización es correcto.

Como pudimos ver en la hoja de recursos, cada uno tiene asignado un coste. Este coste, unido a la duración de las tareas, y el porcentaje de implicación de cada recurso en cada tarea, nos ofrece como resultado el coste total del proyecto, que asciende a cuarenta y nueve mil doscientos veinte euros.

Informe presupuestario

Id	Nombre de tarea	Acumulación de costos fijos	Costo total
10	Arquitectura y estudio del entorno tecnol	Prorrateo	5.550,00 €
18	Control de errores	Prorrateo	608,00 €
2	Descripción general	Prorrateo	3.520,00 €
6	Diagrama de casos de uso	Prorrateo	440,00 €
11	Diagramas de secuencia	Prorrateo	2.250,00 €
12	Diseño de la estructura de datos	Prorrateo	680,00 €
14	Diseño de plan de pruebas	Prorrateo	950,00 €
13	Diseño del flujo de datos	Prorrateo	760,00 €
15	Diseño del interfaz de usuario y servidor	Prorrateo	420,00 €
7	Especificación de casos de uso	Prorrateo	1.200,00 €
8	Especificación del flujo de mensajes	Prorrateo	450,00 €
1	Estudio de requisitos	Prorrateo	5.280,00 €
21	Funcionalidades del sistema	Prorrateo	8.460,00 €
4	Identificación de requisitos	Prorrateo	1.400,00 €
22	Implementación de interfaz de usuario	Prorrateo	280,00 €
17	Mecanismos de crifrado y seguridad	Prorrateo	760,00 €
24	ordenadores	Prorrateo	1.800,00 €
23	Plan de pruebas globales	Prorrateo	10.080,00 €
20	Protocolo de comunicación	Prorrateo	1.792,00 €
19	Protocolo de ejecución de órdenes	Prorrateo	680,00 €
26	Redes	Prorrateo	400,00 €
5	Roles del sistema	Prorrateo	660,00 €
25	Servidor	Prorrateo	800,00 €
			49.220,00 €

5.2. CICLO DE VIDA

Durante el desarrollo de todo proyecto software, desde que se detecta la necesidad de desarrollar el mismo hasta que éste es retirado, existe la necesidad de dividir el proyecto en módulos más pequeños llamados "etapas". Dependiendo de las necesidades del proyecto, las etapas se organizan de una forma determinada para intentar optimizar el desarrollo del proyecto y adecuarlo a la realidad específica del producto. Esta organización la definimos como "ciclo de vida software".

Las formas de organizar y estructurar la secuencia de ejecución de las tareas en las diferentes fases del proyecto da lugar a diferentes definiciones de "ciclos de vida". Existen multitud de clasificaciones del ciclo de vida del desarrollo software según autores, paradigmas de programación y necesidades del proyecto, y cada uno tiene sus ventajas e inconvenientes, pero entre los más destacados podemos nombrar el "ciclo de vida en cascada", el "ciclo de vida en V", o el "ciclo de vida iterativo incremental". Lo que buscamos con la elección de un ciclo de vida concreto es el control de cada etapa de forma sistemática y eficiente, pudiendo así obtener un producto libre de errores y optimizado en coste.

La realidad del proyecto a realizar en nuestro caso manifiesta los siguientes factores:

- Se basa en una plataforma de desarrollo nueva para los programadores, y el desarrollo es complejo, con lo que el aprendizaje en el transcurso de la realización del proyecto es continuo, y frecuentemente se descubren nuevos mecanismos para optimizar ciertas partes del desarrollo ya realizado.
- Es un desarrollo complejo, en el que las pruebas son constantes, y la retroalimentación con la información recibida del usuario es clave para el desarrollo. Esta retroalimentación obliga a veces a redefinir partes del producto.
- Aunque a menudo infravalorado, las necesidades del usuario pueden redefinir ciertas características del producto, y por tanto de los requisitos, que necesitan ser refinados en sucesivas fases. En un ciclo de vida con “etapas” cerradas puede suponer un problema.

Por todos estos factores se elige un ciclo de vida de desarrollo "iterativo incremental". Este ciclo de vida se ajusta a nuestro desarrollo, y nos ofrece las siguientes ventajas:

- Se divide el desarrollo en "pequeños ciclos", que afinan y redefinen posteriores ciclos del desarrollo, optimizando el análisis de los mismos, y produciendo un desarrollo más rápido y con menos posibilidad de errores.
- La división del trabajo en pequeños "ciclos" hace más sencillo su finalización al tener objetivos más pequeños y definidos.
- Se reparte mejor la carga de trabajo entre los diferentes recursos al realizarse tareas de forma paralela, quedando ocupados mayor porcentaje de tiempo.
- Si esto lo unimos a la retroalimentación con la información proporcionada por anteriores ciclos, y la disminución de errores que esto provoca, obtenemos también una reducción del tiempo y coste de ejecución del proyecto.

6. CONCLUSIONES

6.1. SOBRE EL TRABAJO REALIZADO

Tras la conclusión del proyecto fin de carrera, y del presente documento, se pueden extraer muchas y positivas conclusiones.

La primera parte del proyecto era la familiarización con una herramienta moderna de desarrollo, como es todo el entorno de .NET, sus librerías, sus clases, y todo su potencial. Además, esta familiarización incluía el aprendizaje de un nuevo lenguaje de programación, Visual Basic. NET, y una nueva forma de programar, menos estructurada y más orientada a objetos. Todo este esfuerzo de aprendizaje concluye con un manual (el anexo de este documento) que podrá servir de referencia a cualquier persona que quiera iniciarse en este entorno con este lenguaje. Pero esta labor instructiva no ha consistido únicamente en la realización del manual: se han leído cientos de documentos y paginas de internet, ha habido una intensa labor de consulta en foros como los de MSDN, se han creado decenas de pequeños programas de prueba testeando las distintas posibilidades que ofrece el programa (por ejemplo se ha creado un servidor para un juego online, un programa para sincronizar unidades de almacenamiento, un programa que genera estadísticas a partir de logs, otro programa de búsqueda de "cadenas de texto" dentro de ficheros…). Una labor intensa, pero gratificante. De hecho, gracias a los conocimientos adquiridos, pude entrar en una consultoría en la que el trabajo diario se basa en este lenguaje.

La segunda etapa del proyecto ha consistido en realizar un programa P2P desde cero, incluyendo la parte de cliente y la parte de servidor, aplicando todos los conocimientos de la carrera, y los aprendidos para el propio proyecto. Como el sistema es un nuevo concepto en el mundo de los P2P, ha sido necesaria una considerable fase de planificación de los distintos elementos (espacio compartido, cifrado, conexiones, etc) para que a la hora de implementar el software, no se de un paso adelante, y dos atrás.

Una vez realizada la planificación se pasó a implementar el software; dicho trabajo ha sido bastante complejo, ya que no se trata de un programa de ejecución lineal, fácilmente implementable y fácil a la hora de la detección de errores: es un programa que usa conexiones a través de socket, ha habido que coordinar el

desarrollo de cliente y servidor paralelamente para que uno diera respuestas a la necesidad del otro, hay una orientación a objetos en el desarrollo del software que es un nuevo concepto respecto a lo aprendido en la facultad, hay ejecución mediante "hilos" que dificulta la programación y detección de errores, la fase de prueba es realmente compleja ya que se trata de "una red" de ordenadores y no de un único terminal...

Además, se ha querido tocar las distintas áreas de conocimiento aprendidas durante la carrera: se han usado sockets y se ha creado un protocolo propio de comunicación para la transmisión de mensajes para adecuar el desarrollo del proyecto lo aprendido en asignaturas como Sistemas Operativos, se ha usado ejecución mediante hilos, se ha cifrado la información, se ha usado un hash de detección de errores, se ha hecho una planificación del proyecto, se han usado distintos tipos diagramas como diagramas de actividad o casos de uso, muchos elementos de programación como el acceso a ficheros, etc. En definitiva, un repaso por numerosas áreas de conocimiento impartidas en la carrera.

Todos estos elementos combinados con mucho tiempo y esfuerzo dan como resultado el trabajo aquí presente: el aprendizaje de una nueva herramienta de trabajo, y de un nuevo lenguaje de programación, el desarrollo de un prototipo funcional, y la realización de una memoria de dicho trabajo.

El único punto negativo a destacar, más quizás por la finalidad de la realización de un "proyecto fin de carrera de una Ingeniería Técnica" que como punto negativo en sí, es, precisamente, la magnitud del proyecto aquí presente, muy amplio en todos los aspectos, que ha llevado mucho esfuerzo y tiempo, quizás más esfuerzo y tiempo del que hubiera sido aconsejable. Una mejor acotación de los límites del proyecto hubiera solucionado este inconveniente.

6.2. VISIÓN DE FUTURO

Realizar un software P2P de calidad, estable, y profundizando en todas las áreas, depurando el programa, corrigiendo errores… es un trabajo que lleva años a equipos de desarrollo de muchas personas . Como hemos comentado durante todo el documento, se ha intentado tocar las diferentes áreas de conocimiento adquirido durante la carrera para implementar el software, sin entrar excesivamente en profundidad en ellos ya que no era la finalidad del proyecto. Lo que aquí tenemos es un "prototipo funcional" pero, un trabajo de futuro podría ser, obviamente, profundizar en todas estas áreas:

- Seguridad: Mejorar el mecanismo de cifrado con algún algoritmo moderno de cifrado, muchos de los cuales suministra las propias librerías de .NET.
- Hash: Como hemos comentado durante este documento, se ha implementado un pequeño hash para la detección de errores. Este hash se puede ampliar y mejorar, e incluso de podría incluir un hash como el que posee el Emule, que no solo detecta errores sino que los corrige.
- Mejorar la orientación a objetos: La orientación a objetos es un sistema de programación totalmente ajeno a los conocimientos prácticos adquiridos en la carrera. Su realización ha sido costosa y no en todos los casos de ha podido realizar, así que un área en el que trabajar es en la ampliación de la orientación a objetos del software.
- Mejorar el concepto: El nuevo concepto de P2P implica múltiples elementos que pudieran ser susceptibles de mejoras en el propio concepto en sí. Podría intentar desligar la dependencia de los clientes de un único servidor y hacer la red descentralizada, o cuando menos una red mixta. Podría disminuir la carga de trabajo del servidor con las cabeceras de los trozos de archivo, haciendo que fueran los propios clientes quienes manejaran esa información, intentando conservar la seguridad de la red. Podría hacerse un estudio del tamaño de los ficheros para encontrar tamaños más óptimos de trozos de archivo.
- Testeo real de la red: Uno de los elementos más complejos a la hora de desarrollar el software es la detección y corrección de fallos. Para hacerlo correctamente se necesitaría un grupo de usuarios en distintos ordenadores probando durante meses el software, unido a un sistema recolector de informes. Algo muy difícil cuando hablamos de un solo usuario desarrollando y probando el programa.

7. REFERENCIAS

Relacionado con .NET y VB.NET.

1. EVJEN, Hill. *Profesional Visual Basic 2005*. Indianapolis, Indiana: Wiley Publishing, Inc., 2006. ISBN-13: 978-07645-7536-5. ISBN-10: 0-7645-7536-8.

2. SOM, Guillermo. *Curso de iniciación a la programación con Visual Basic .NET*. [en línea]. Septiembre 2001 (última revisión abril 2007). <http://www.elguille.info/NET/cursoVB.NET/default.aspx>.

3. ISO. *Information technology — Common Language Infrastructure (CLI) Partitions I to VI* [en línea]. ISO/IEC 23271:2006. ISO/IEC 2006. <http://standards.iso.org/ittf/PubliclyAvailableStandards/c042927_ISO_IEC_23271_2006(E).zip>. Fuente: <http://msdn.microsoft.com/en-us/netframework/aa569283.aspx>.

4. *Lo nuevo de .NET Framework versión 3.5* [en línea]. Microsoft Corporation: noviembre 2007. <http://msdn.microsoft.com/es-es/library/bb332048.aspx>

5. RICHTER, Jeffrey. *Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework* (MSDN Magazine) [en línea]. Microsoft Corporation: noviembre 2000. <http://web.archive.org/web/20070703083608/http://msdn.microsoft.com/msdnmag/issues/1100/GCI/>

6. GRIMES, Richard ..*NET Delegates: Making Asynchronous Method Calls in the .NET Environment*. (MSDN Magazine) [en línea]. Microsoft Corporation: agosto 2001. <http://msdn.microsoft.com/es-es/magazine/cc301332%28en-us%29.aspx>

7. *Programming the Thread Pool in the .NET Framework*. [en línea]. Microsoft Corporation: junio 2002. <http://msdn.microsoft.com/en-us/library/ms973903.aspx>

8. *Comunidad de bibliotecas de clases base*. [en línea]. Microsoft Corporation. <http://msdn.microsoft.com/es-es/netframework/aa569603.aspx >

Relacionado con "Internet", "P2P", "redes", "Hash", "Protocolos de redes", "Sistemas distribuidos", etc.

9. WALDEMAR HERLITZ GATICA, Heinz. *Transversabilidad en NAT/Firewall*. [en línea]. Temuco: 2005. Cap. 4 (relacionado con P2P). <http://www.uct.cl/biblioteca/tesis-on-line/heinz-herlitz/tesis.pdf >

10. ANDROUTSELLIS-THEOTOKIS, Stephanos. SPINELLIS, Diomidis. *A Survey of Peer-to-Peer Content Distribution Technologies*. [en línea] ACM Computing Surveys. December 2004, núm. 36, p. 335-371. <http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html>

11. BIDDLE, Peter. ENGLAND, Paul. PEINADO, Marcus. WILLMAN, Bryan. *The Darknet and the Future of Content Distribution*. [en línea]. ACM Workshop on Digital Rights Management: noviembre 2002. <http://crypto.stanford.edu/DRM2002/darknet5.doc>

12. SCHODER, Detlef. FISCHBACH, Kai. SCHMITT, Christian. *Core Concepts in Peer-to-Peer Networking* [en línea]. Idea Group Inc: 2005. <http://www.idea-group.com/downloads/excerpts/Subramanian01.pdf >

13. STEWART, William. *Internet History*. [en línea]. Living Internet Inc. Ottawa: 1996-2009. <http://www.livinginternet.com/i/ii.htm>

14. WALDROP, Mith. *DARPA and the Internet Revolution*. [en línea]. Defense Advanced Research Projects Agency. <http://www.darpa.mil/Docs/Internet_Development_200807180909255.pdf> y <http://www.darpa.mil/internetbibliography.html >

15. HURTADO JARA, Omar. *Sistemas distribuidos*. [en línea]. Universidad Carlos III De Madrid. <http://www.monografias.com/trabajos16/sistemas-distribuidos/sistemas-distribuidos.shtml>

16. BRAVO, David. *Redes P2P y propiedad intelectual*. [en linea]. 802 Party: 2005. [VIDEO]. <http://video.google.com/videoplay?docid=-7012884909062999701>

17. R. Schollmeier. *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*. First International Conference on Peer-to-Peer Computing (P2P'01), 2001. ISBN: 0-7695-1503-7.

18. MANÍCULA. *Gestión de la corrupción: ICH y AICH*. [en línea]. Julio 2009. <http://www.emule-configurar.com/2009/07/gestion-de-la-corrupcion-ich-y-aich.html>

19. S. TANENBAUM, Andrew. VAN RENESSE, Robbert. *Distributed Operating Systems*. [en línea] Computing Surveys, Vol. 17, Núm. 4. December 1985. <http://cactus.eas.asu.edu/Partha/Teaching/531-common-files/p419-tanenbaum.pdf>

20. *Windows Peer-to-Peer Networking*. [en línea].]. Microsoft Corporation. <http://technet.microsoft.com/en-us/network/bb545868.aspx>

21. ROJO, Oscar. *Introducción a los sistemas distribuidos*. [en línea]. 2003. <http://www.augcyl.org/?q=glol-intro-sistemas-distribuidos>

22. BUFORD, John. YU, Heather. *P2P Networking And Applications*. Morgan Kauffman: diciembre 2008. ISBN-13: 978-0-12-374214-8.

23. GARCÍA CARBALLEIRA, Félix. *Máster en Ciencia y Tecnología Informática*. [en línea]. Curso 2006-2007. <http://arcos.inf.uc3m.es/~fgarcia/sd/Introduccion.pdf>

24. EARLE MOORE, Gordon. *Cramming more components onto integrated circuits*. [en línea]. Electronics, Volumen 38, núm. 8: abril 1965. <ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf>

25. *Redes P2P y protocolo*. [en línea]. 14 de diciembre de 2008. <http://beetonix.com/post/tecnologia/redes-p2p-y-protocolo/>

26. DUNCAN, Ralph. *A Survey of Parallel Computer Architectures*. [en línea]. IEEE Computer Society Press. Los Alamitos, USA: Febrero 1990. <http://cs.nju.edu.cn/~gchen/teaching/fpc/Duncan90.pdf>

27. BURATTO, Carina. CANAPARO, Ana Laura. LABORDE, Andrea. MINELLI, Alejandra. *La informática como Recurso Pedagógico-Didáctico en la Educación*. [en línea]. <http://www.monografias.com/trabajos10/recped/recped.shtml>

28. PARDO, Lisandro. *Alcatel rompe récord: 100 petabits por segundo*. [en línea]. NeoTeo: 01 de octubre de 2009. <http://www.neoteo.com/alcatel-rompe-record-100-petabits-por-segundo.neo>

29. MUÑOZ, Ramón. *Un millón de hogares enganchados a eMule*. [en línea]. El País. Madrid: 23/10/2006. <http://www.elpais.com/articulo/sociedad/millon/hogares/enganchados/eMule/elpporsoc/20061023elpepisoc_1/Tes>

30. GHODSI, Ali. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. [en línea]. Royal Institute of Technology. Stockholm, Sweden: Octubre 2006. ISSN 1653-6363. <http://www.sics.se/~ali/thesis/dks-original.pdf>.

ANEXO A

1. INTRODUCCIÓN A LA PLATAFORMA .NET

1.1. DESCRIPCIÓN.

Éste es un manual de introducción al lenguaje VB.net destinado a gente con ciertos conocimientos de programación para que le pueda servir de guía a la hora de introducirse en este nuevo lenguaje. No se pretenden definir todos los conceptos comunes a los lenguajes de programación, si no cómo usar esos elementos directamente con ejemplos prácticos (que irán en cursiva), que toda persona con ciertos conocimientos de programación conoce, tales como variables, bucles, definición de tipos, estructuras, etc.

1.2. ¿POR QUÉ VISUAL BASIC. NET?

Todos los lenguajes ".NET" tienen acceso a las mismas bibliotecas de clases y funciones, y todos ellos hacen previamente a la ejecución un precompilado en un lenguaje intermedio, con lo que en definitiva lo que cambia de uno a otro es la sintaxis, pero todos, excepto funcionalidades muy específicas, tienen las mismas posibilidades y rendimientos similares. Elegimos Visual Basic .NET porque es un lenguaje muy extendido del que tenemos cientos de manuales de referencia, foros, ejemplos, y demás, con lo que nos facilita mucho la labor de aprendizaje del mismo. Es muy aconsejable visitar la página española del "MSDN" donde están documentadas todas las librerías, todas las funciones, con ejemplos prácticos en todos los lenguajes .Net.

PÁGINA: http://msdn2.microsoft.com/es-es/

1.3. EL ENTORNO DE EJECUCIÓN CLR

Common Language Runtime (CLR) es la implementación de Microsoft de un estándar llamado Common Language Infrastructure o CLI. El CLR/CLI

esencialmente define un entorno de ejecución virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET.

1.4. EL LENGUAJE INTERMEDIO Y EL CLS

Dentro de la CLI, existe un lenguaje llamado IL (Intermediate Language o Lenguaje Intermedio). Cuando se compila una aplicación escrita en un lenguaje .NET cualquiera, el compilador lo que genera en realidad es un nuevo código escrito en este lenguaje intermedio.

Para conseguir la interoperabilidad entre lenguajes no sólo llega con el lenguaje intermedio, sino que es necesario disponer de unas "reglas del juego" que definan un conjunto de características que todos los lenguajes deben incorporar. A este conjunto regulador se le denomina Common Language Specification (CLS) o, en castellano, especificación común de los lenguajes.

1.5. APLICACIONES MODO CONSOLA

Las aplicaciones en modo Consola son las aplicaciones básicas .NET en las que el programa para ejecutarse abre una Consola (ventana de MS-DOS). En la mayoría de las universidades trabajan con este tipo de lenguajes ya que son bastante simples a la hora del aprendizaje, por eso mismo, para dar un paso “mas allá” este manual está enfocado a las aplicaciones con ventanas.

1.6. APLICACIONES WINDOWS FORMS

Las aplicaciones de escritorio son aquellas basadas en ventanas y controles comunes de Windows que se ejecutan en local. Son de las que en principio hablaremos y en las que haremos la mayoría de pruebas del lenguaje.

1.7. APLICACIONES WEB FORMS

En el modelo orientado a eventos se define la interfaz de usuario colocando controles en un contenedor y se escribe el código que actuará como respuesta a las interacciones de los usuarios sobre estos controles.

2. CARACTERÍSTICAS DEL LENGUAJE VB.NET

2.1. EL SISTEMA DE TIPOS

Tipos primitivos

El lenguaje intermedio tiene una definición de tipos, común a todos los lenguajes. En cada lenguaje la sintaxis cambia, pero una vez hecho el "compilado" del programa, se transforman en esos tipos. A la izquierda esta el tipo primitivo, a la derecha el tipo en VB .NET.

.Net Framework	Visual Basic .NET
System.Boolean	Boolean
System.Byte	Byte
System.Int16	Short
System.Int32	Integer
System.Int64	Long
System.Single	Single
System.Double	Double
System.Decimal	Decimal
System.Char	Char
System.String	String
System.Object	Object
System.DateTime	Date
System.SByte	SByte
System.UInt16	UShort
System.UInt32	UInteger
System.UInt64	ULong

Sufijos o caracteres y símbolos identificadores para los tipos

Cuando usamos valores literales numéricos en Visual Basic 2005, el tipo de datos que por defecto le asigna el compilador es el tipo Double. Si nuestra intención es utilizar sin embargo un tipo de datos diferente, podemos indicarlo añadiendo una letra como sufijo al tipo. Por ejemplo para indicar que queremos considerar un valor entero podemos usar la letra I o el signo %.

Tipo de datos	Símbolo	Carácter
Short	N.A.	S
Integer	%	L
Single	!	F
Double	#	R
Decimal	@	D
UShort	N.A.	US
UInteger	N.A.	UI
ULong	N.A.	UL

Tipos por valor y tipos por referencia

Los tipos por valor son tipos de datos cuyo valor se almacena en la pila o en la memoria "cercana".

Los tipos por referencia se almacenan en el "montículo" (heap) o memoria "lejana". A diferencia de los tipos por valor, los tipos por referencia lo único que almacenan es una referencia (o puntero) al valor asignado

* Declarar variables:
Dim a As Integer
Dim a, b, c As Integer

* Declarar variables y asignar el valor inicial:
Dim a As Integer = 10
Dim a As Integer = 10, b As Integer = 25

* El tipo de datos Char
Dim c As String
Dim c As Char = "A"c

* Asignar un determinado valor ASCII a una variable de tipo “Char”:
c = Chr(65)

* Asignar a una variable tipo Integer un el valor ASCII correspondiente a un Char:
Dim n As Integer
n = Asc("A")

* Declarar constantes:
Const numero as Integer = 10

* Otras declaraciones de tipos
Dim j, k As Integer, s1, Nombre As String, d1 As Decimal
Dim z As Object
Dim unaFecha As Date = #10/27/2001#
Dim unDecimal As Decimal = 99912581258.125D
Dim unDecimal2 As Decimal = 9876543210123456@
Dim unDoble As Double = 125897.12045R
Dim unDoble2 As Double = 2457998778745.4512#
Dim unEntero As Integer = 250009I
Dim unEntero2 As Integer = 652000%
Dim unLong As Long = 123456789L
Dim unLong2 As Long = 987654&
Dim unShort As Short = 32000S

* Conversiones de tipos: Operaciones para convertir de unos tipos a otros.

Nombre de la función	**Tipo de datos que devuelve**	**Valores del argumento "expresion"**
CBool(*expresion*)	Boolean	Cualquier valor de cadena o expresión numérica.
CByte(*expresion*)	Byte	de 0 a 255; las fracciones se redondean.
CChar(*expresion*)	Char	Cualquier expresión de cadena; los valores deben ser de 0 a 65535.
CDate(*expresion*)	Date	Cualquier representación válida de una fecha o una hora.
CDbl(*expresion*)	Double	Cualquier valor Duoble, ver la tabla anterior para los valores posibles.
CDec(*expresion*)	Decimal	Cualquier valor Decimal, ver la tabla anterior para los valores posibles.
CInt(*expresion*)	Integer	Cualquier valor Integer, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CLng(*expresión*)	Long	Cualquier valor Long, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CObj(*expresion*)	Object	Cualquier expresión válida.
CShort(*expresion*)	Short	Cualquier valor Short, ver la tabla anterior para los valores posibles, las fracciones se redondean.
CSng(*expresion*)	Single	Cualquier valor Single, ver la tabla anterior para los valores posibles.
CStr(*expresion*)	String	Depende del tipo de datos de la expresión.
		Nota: Todos los objetos de vb.NET tienen unos métodos para realizar conversiones a otros tipos, al menos de número a cadena, ya que tienen la propiedad .ToString que devuelve una representación en formato cadena del número en cuestión (igual que CStr).
CType(*expresion*, Tipo)	El indicado en el segundo parámetro	Cualquier tipo de datos

Val(*expresion*)	Double	Una cadena de caracteres.
Fix(*expresion*)	Depende del tipo de datos de la expresión	Cualquier tipo de datos. devuelve un valor del *mismo tipo* que el que se indica en el parámetro o expresión, pero sin decimales. si esos números son negativos, Fix devuelve el siguiente valor igual o mayor que el número indicado
Int(*expresion*)	Depende del tipo de datos de la expresión	Cualquier tipo de datos. devuelve un valor del *mismo tipo* que el que se indica en el parámetro o expresión, pero sin decimales. (igual a Fix, se supone). si esos números son negativos, Int lo hace con el primer número menor o igual...

Enumeraciones: Constantes agrupadas

Una enumeración es un grupo de constantes que están relacionadas entre sí.

* Definir un tipo enumerado:

Enum Colores
  Rojo
  Verde
  Azul
End Enum

Nota: El tipo de datos de cada miembro de la enumeración será Integer al no tener tipo

Enum Colores As Long
  Rojo
  Verde
  Azul
End Enum

Nota: El tipo de datos de cada miembro de la enumeración será Long

<Flags()> _
Enum Colores As Byte

```
   Rojo = 1
   Verde = 2
   Azul = 4
End Enum
```

Nota: De esta forma podremos indicar unos valores concretos para los miembros de la enumeración. Se usa también para indicar valores que se pueden "sumar" o complementar entre sí, pero sin perder el nombre

* Sumar valores de enumeraciones a las que hemos aplicado el atributo Flags y a las que no se lo hemos aplicado:

```
Enum Colores As Byte
   Rojo = 1
   Verde = 2
   Azul = 4
End Enum
Dim c As Colores = Colores.Azul Or Colores.Rojo
Dim s As String = c.ToString
```

Nota: El contenido de la variable s será "5".

```
<Flags()> _
Enum Colores As Byte
   Rojo = 1
   Verde = 2
   Azul = 4
End Enum
Dim c As Colores = Colores.Azul Or Colores.Rojo
Dim s As String = c.ToString
```

Nota: El contenido de la variable s será: "Rojo, Azul".

Arrays (matrices)

* Declaración de arrays:

```
Dim nombres( ) As String
Dim nombres(10) As String
Dim nombres(0 To 10) As String
```

* Declarar e inicializar un array:
Dim nombres( ) As String = {"Pepe", "Juan", "Luisa"}
Dim nombres(,) As String = {{"Juan", "Pepe"}, {"Ana", "Eva"}}

* Cambiar el tamaño de un array
Dim nombres(10) as String
Redim nombres(12)
Nota: Con esto hemos cambiado el tamaño del array de 10 a 12, pero hemos perdido todos los datos que contuvieran sus elementos.

* Cambiar el tamaño de un array y mantener los valores que tuviera anteriormente:
Redim preserve nombres (12)
Nota: Si bien tanto ReDim como ReDim Preserve se pueden usar en arrays de cualquier número de dimensiones, en los arrays de más de una dimensión solamente podemos cambiar el tamaño de la última dimensión.

* Formas de eliminar el contenido de un array: redimensionando el array indicando que tiene cero elementos (aunque siempre tendria 1 como minimo), usar la instrucción “erase”, asignar un valor “Nothing” al array…

* Los arrays son tipos por referencia:
Dim nombres() As String = {"Juan", "Pepe", "Ana", "Eva"}
Dim otros() As String
otros = nombres
nombres(0) = "Antonio"
Nota: Con esto modificamos nombres(0) pero tambien otros(0).

Dim nombres() As String = {"Juan", "Pepe", "Ana", "Eva"}
Dim otros() As String
ReDim otros(nombres.Length)
nombres.CopyTo(otros, 0)
nombres(0) = "Antonio"
Nota: Con CopyTo hacemos una copia de los elementos del array. El array de destino tiene que tener tamaño suficiente para contener el de origen.

* Para averiguar el número de elementos de un array, también podemos usar la función UBound., Length

Dim otros(5) As String
Dim n as integer
n = otros.length
Nota: n = 6, ya que el primer elemento del array es “0”

* Los arrays multidimensionales
Dim a2(,) As Integer
Dim a3(, ,) As Integer
Dim b2(1, 6) As Integer
Dim b3(3, 1, 5, 2) As Integer
Dim c2(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
Dim c3(, ,) As Integer = {{{1, 2}, {3, 4}, {5, 6}}, _
{{7, 8}, {9, 10}, {11, 12}}, _
{{13, 14}, {15, 16}, {17, 18}}, _
{{19, 20}, {21, 22}, {23, 24}}}
NOTA: En el último caso, he usado el continuador de líneas para que sea más fácil "deducir" el contenido

2.2. OPERADORES, CONDICIONALES, BUCLES, Y OTROS MECANISMOS DE CONTROL.

Prioridad de los operadores aritméticos y de concatenación

Exponenciación (^), Negación (-), Multiplicación y división (*, /), División de números enteros (\), Módulo aritmético (Mod), Suma y resta (+, -), Concatenación de cadenas (&).

NOTA: si tienen la misma prioridad se evalúan de izquierda a derecha, por ejemplo: 15 - 3 - 4

Operadores de comparación

= igual
< menor que
> mayor que
<= menor o igual
>= mayor o igual
<> distinto

Operadores logicos

* Los operadores lógicos fundamentales: AND (conjunción), OR (disyunción), NOT (negación).

* Otros operadores lógicos: ANDALSO, ORELSE, XOR
IF N = 3 ANDALSO X > 10 THEN
NOTA: si no se cumple n=3 no comprueba lo siguiente. Si solo pusieras "and" si lo comprueba, y no es necesario.

IF N = 3 ORELSE X > 10 THEN
NOTA: Si la primera condición, n=3 se cumple, no comprueba la segunda.

Sentencia "if"

If a = 10 Then
ElseIf a = 15 Then
Else
End If

Bucles

* For
For <variable numérica> = <valor inicial> To <valor final> [Step <incremento>]
[código]
Next

* For each: ejecuta una vez "código" por cada elemento en la "colección de elementos"
For Each <variable> In <colección del tipo de la variable>
[código]
Next

* While <expresión>
[código]
End While

* Do While <expresión>
[código]
Loop

* Do
[código]
Loop While <expresión>

* Do Until <expresión>
[código]
Loop

* Do
[código]
Loop Until <expresión>

NOTA: Para salir del los bucles si esperar a que acaben se pueden usar las ordenes "*Exit For*", "*Exit While*", "*Exit Do*".

Sentencia Case

```
Select Case <expresión a evaluar>
  Case <lista de expresiones>
  Case <otra lista de expresiones>
  Case Else  ' si no se cumple ninguna de las listas de expresiones
End Select
```

Ejemplo:

```
Select Case i
    Case 3
    Case 6 To 11
    Case 14, 17
    Case Is > 25
    Case Else
End Select
```

Captura de errores: Try

```
Try    ' el código que puede producir error
  [código]
Catch [tipo de error a capturar]    ' código cuando se produzca un error
  [código]
Finally    ' código se produzca o no un error
  [código]
End Try
```

"Catch" y "finally" no son obligatorios, pero al menos hay que usar uno de ellos. Si decidimos "prevenir" que se produzca un error, pero simplemente queremos que el programa continúe su ejecución, podemos usar un bloque Catch que esté vacío. Podemos usar más de un bloque Catch, si es que nuestra intención es detectar diferentes tipos de errores.

Si se produce una excepción o error en un bloque Try, el código que siga a la línea que ha producido el error, deja de ejecutarse, para pasar a ejecutar el código que

haya en el bloque Catch, se seguiría (si lo hubiera) por el bloque Finally y por último con lo que haya después de End Try.

Si no se produce ningún error, se continuaría con todo el código que haya en el bloque Try y después se seguiría (si lo hubiera), con el bloque Finally y por último con lo que haya después de End Try.

```
Dim i, j As Integer
Try
   i = 10
   j = 0
   i = i \ j
Catch
   ' no hacemos nada si se produce un error
End Try
Catch ex As DivideByZeroException
   Console.WriteLine("ERROR: división por cero")
Catch ex As OverflowException
   Console.WriteLine("ERROR: de desbordamiento (número demasiado grande)")
Catch ex As Exception
   Console.WriteLine("Se ha producido el error: {0}", ex.Message)
End Try
```

NOTA: Hay que tener cuidado de poner el tipo genérico al final

2.3. LOS PROCEDIMIENTOS Y LAS FUNCIONES

Ejemplo de procedimiento y ejemplo de función

```
Module Module1
   Sub Main()
      MostrarS()
      Dim s As String = MostrarF()
      Console.WriteLine(s)
   End Sub

   Sub MostrarS( )
```

```
    Console.WriteLine("Este es el procedimiento MostrarS")
  End Sub

  Function MostrarF( ) As String
    Return "Esta es la función MostrarF"
  End Function
End Module
```

NOTA: “return” indica el punto de retorno de la función. En este caso la función debe devolver un string, y el tipo del parámetro devuelto por “return” tiene que ser, y es, de tipo “string”

Parámetros o argumentos de los procedimientos.

```
Sub Saludar (ByVal tipoSaludo As String, ByVal nombre As String)
  Console.WriteLine(tipoSaludo & " " & nombre)
End Sub
```

NOTA: Byval indica el parámetro “por valor”.

```
Sub Saludar3(ByRef nombre As String)
  nombre = "Hola " & nombre
  Console.WriteLine(nombre)
End Sub
```

NOTA: ByRef indica que es una variable “por referencia”.

Parámetros opcionales

```
Sub Prueba(ByVal uno As Integer, Optional ByVal dos As Integer = 5)
...
End sub
```

Con esta declaración podemos usar este procedimiento de estas dos formas: Prueba(10, 20) o Prueba(10). Si no indicamos el segundo parámetro, el valor que se usará dentro del procedimiento será el valor indicado en la declaración.

Sobrecarga de procedimientos

Sub Prueba (ByVal uno As Integer)
Sub Prueba(ByVal uno As Integer, ByVal dos As Integer).

Dos declaraciones de procedimientos distintos, con mismo nombre, pero distintos parámetros. Se podría llamar a Prueba (1) o Prueba (1,1) indistintamente y ejecutaría una u otra dependiendo del numero y forma de parámetros.

Array de parámetros opcionales

Puede ser que necesitemos que un procedimiento reciba una cantidad no predeterminada de parámetros, en esos casos podemos usar un array (o matriz) de parámetros opcionales.

Sub Prueba3(ByVal uno As Integer, ByVal ParamArray otros( ) As Integer)

Podemos llamarlo usando uno o más parámetros:Prueba3(1) o Prueba3(1, 2, 3, 4, 5, 6, 7) o tb Prueba3(1, 2). No es necesario usar la palabra "Optional", ya que estos parámetros son opcionales de forma predeterminada.

3. CLASES Y ESTRUCTURAS

3.1. CLASES: TIPOS POR REFERENCIA DEFINIDOS POR EL USUARIO

Los tipos por referencia realmente son clases, de la cuales debemos crear una instancia para poder usarlas.

La herencia: Característica principal de la "programación orientada a objetos". Característica que nos permite ampliar la funcionalidad de una clase existente sin perder la que ya tuviera previamente. Esta nueva clase puede cambiar el comportamiento de la clase base (de la cual hereda) y/o ampliarlo, de esta forma podemos adaptar la clase original a nuestras necesidades.

La encapsulación nos permite abstraer la forma que tiene de actuar una clase sobre los datos que contiene o manipula.

El polimorfismo es una característica que nos permite realizar ciertas acciones o acceder a la información de los datos contenidos en una clase de forma semi-anónima, al menos en el sentido de que no tenemos porqué saber sobre que tipo objeto realizamos la acción.

"Object": La clase base de todas las clases de .NET

Nota: Todas las clases de .NET se derivan siempre de forma automática de la clase "Object".

Definir una clase

```
Class Cliente
   Public Nombre As String
   Public Apellidos As String
   Sub Mostrar()
      Textbox1.text = "El nombre del cliente:" & Nombre
   End Sub
End Class
```

Los miembros de una clase

Enumeraciones, Campos (variables usadas para mantener los datos que la clase manipulará), Métodos (funciones o procedimientos), Propiedades ("características" de las clases y la forma de acceder "públicamente" a los datos que contiene), Eventos (mensajes que la clase puede enviar para informar que algo está ocurriendo en la clase).

Crear un objeto a partir de una clase

```
Dim cli As Cliente
```

cli = New Cliente( )

O directamente:
Dim cli As New Cliente( )

Acceder a los miembros de una clase

cli.Nombre = "Guillermo" 'definimos un valor
cli.Mostrar( ) 'llamamos a un metodo de la clase

Usar la herencia. "Inherits"

```
Class Cliente
   Public Nombre As String
   Sub Mostrar()
      Textbox1.text = "El nombre del cliente:" & Nombre
   End Sub
End Class

Class ClienteMoroso
   Inherits Cliente
   Public Deuda As Decimal
End Class
```

Polimorfismo

```
Dim cli As Cliente
Dim cliM As New ClienteMoroso( )
cliM.Nombre = "Pepe"
cliM.Deuda = 2000
cli = cliM
```

NOTA: La variable "cli" simplemente se ha declarado como del tipo Cliente, pero no se ha creado un nuevo objeto, simplemente hemos asignado a esa variable el contenido de la variable "cliM". Cliente "no entiende" nada de las nuevas

propiedades implementadas en la clase derivada, por tanto sólo se podrá acceder a la parte que es común a esas dos clases: la parte heredada de la clase Cliente.

3.2. PROPIEDADES

Imaginémonos que no se debería permitir que se asigne una cadena vacía al Nombre o al Apellido, en el ejemplo anterior, esto no sería posible de "controlar", ya que al ser "variables" públicas no tenemos ningún control sobre cómo ni cuando se asigna el valor. Una propiedad es parecido a una función, pero un poco especial.

```
Public Property Nombre() As String
   ' la parte Get es la que devuelve el valor de la propiedad
   Get
      Return elNombre
   End Get
   ' la parte Set es la que se usa al asignar el nuevo valor
   Set(ByVal Value As String)
      If Value <> "" Then
         elNombre = Value
      End If
   End Set
End Property
```

Value representa el valor que queremos asignar a la propiedad y representará lo que esté a la derecha del signo igual de la asignación. Por ejemplo, si tenemos esto: p.Nombre = "Guillermo", "Guillermo" será lo que Value contenga.

Cuando creamos un procedimiento Property siempre será necesario tener un campo (o variable) privado que sea el que contenga el valor de la propiedad

Para crear una propiedad con el entorno de desarrollo de Visual Studio, ponemos, por ejemplo, “Public Property Nombre() as String”, y el resto (get y set) te lo pone el solo.

Propiedades de sólo lectura

Propiedades a las que no se pueden asignar valores nuevos.

```
Private valorFijo As Integer = 10
Public ReadOnly Property Valor( ) As Integer
   Get
      Return valorFijo
   End Get
End Property
```

Propiedades de sólo escritura.

```
Private valorEscritura As Boolean
Public WriteOnly Property Escribir( ) As Boolean
   Set(ByVal Value As Boolean)
      valorEscritura = Value
   End Set
End Property
```

Campos de sólo lectura

Similar a una constante, pero distinto.

```
Public Class Prueba14_6
   Public ReadOnly LongitudNombre As Integer = 50

   Public Sub New()
   End Sub

   Public Sub New(ByVal nuevaLongitud As Integer)
      LongitudNombre = nuevaLongitud
   End Sub

   Public Shared Sub Main()
```

```
        Dim p As New Prueba14_6()
        ' el valor será el predeterminado: 50
        Console.WriteLine("p.LongitudNombre = {0}", p.LongitudNombre)

        Dim p1 As New Prueba14_6(25)
        ' el valor será el indicado al crear la instancia
        Console.WriteLine("p1.LongitudNombre = {0}", p1.LongitudNombre)
    End Sub
End Class
```

NOTA: Sub Main está declarado como Shared para que se pueda usar como punto de entrada del ejecutable

3.3. EL CONSTRUCTOR: EL PUNTO DE INICIO DE UNA CLASE

Es un método sub llamdo "new". Podríamos dejar el que viene por defecto, o hacer uno de este tipo:

```
Class Cliente
    Public Nombre As String
    Public Apellidos As String
    Public FechaCreacion As Date
    Public Sub New()
        FechaCreacion = Date.Now
    End Sub
End Class
```

Constructores parametrizados

```
Public Sub New()
        FechaCreacion = Date.Now
End Sub

Public Sub New(elNombre As String, losApellidos As String)
        Nombre = elNombre
```

```
        Apellidos = losApellidos
        FechaCreacion = Date.Now
End Sub

Public Sub New (ByVal newNombre as String)
        Me.nombre = newNombre
end sub

Dim c1 As New Cliente
Dim c2 As New Cliente("Jose", "Sánchez")
```

El destructor: El punto final de la vida de una clase

El destructor de Visual Basic 2005 es un método llamado “Finalize”, el cual se hereda de la clase Object. Si asignamos un valor "nothing" a un objeto, se llama automáticamente al Finalice; VB.net se encarga de la destrucción de objetos, en principio no tendríamos por qué ocuparnos nosotros de ello.

3.4. ESTRUCTURAS: TIPOS POR VALOR DEFINIDOS POR EL USUARIO.

Las estructuras pueden contener los mismos miembros que las clases, aunque algunos de ellos se comporten de forma diferente o al menos tengan algunas restricciones, como que los campos definidos en las estructuras no se pueden inicializar al mismo tiempo que se declaran. No es necesario crear una instancia para poder usarlas, ya que es un tipo por valor y los tipos por valor no necesitan ser instanciados para que existan.

Definir una estructura

```
Structure Punto
   Public X As Integer
   Public Y As Integer
End Structure
Dim p As Punto
p.X = 100
p.Y = 75
```

Constructores de las estructuras.

Las estructuras siempre definen un constructor sin parámetros, este constructor no lo podemos definir nosotros, es decir, siempre existe.

```
Structure Punto
   Public X As Integer
   Public Y As Integer
   Sub New(ByVal x As Integer, ByVal y As Integer)
      Me.X = x
      Me.Y = y
   End Sub
End Structure
```

Destructores de las estructuras

En las estructuras no podemos definir destructores, como mucho asignarle nothing.

Campos

Los campos de una estructura no pueden inicialiarse en la declaración y el valor que tendrán inicialmente es un valor "nulo", que en el caso de los campos de tipo numéricos es un cero. Los únicos campos que podemos inicializar al declararlos son los campos compartidos.

3.5. ACCESIBILIDAD Y ÁMBITO

Ámbito

Dependiendo de donde declaremos un miembro o un tipo, éste tendrá mayor alcance o cobertura.

* Ámbito de bloque: Disponible únicamente en el bloque de código en el que se ha declarado. Por ejemplo, si declaramos una variable dentro de un bucle “For” o un “If Then”, esa variable solo estará accesible dentro de ese bloque de código.

```
For i As Integer = 1 To 10...

If n < 5 Then
   Dim j As Integer = n * 3
End If
```

* Ámbito de procedimiento: Disponible únicamente dentro del procedimiento en el que se ha declarado. Cualquier variable declarada dentro de un procedimiento (método o propiedad) solo estará accesible en ese procedimiento y en cualquiera de los bloques internos a ese procedimiento.

```
Function Mostrar() As String
      Dim nombre As String = "Pepita"...
```

* Ámbito de módulo: Disponible en todo el código del módulo, la clase o la estructura donde se ha declarado. Las variables con ámbito a nivel de módulo, también estarán disponibles en los procedimientos declarados en el módulo (clase o estructura) y por extensión a cualquier bloque dentro de cada procedimiento.

* Ámbito de espacio de nombres: Disponible en todo el código del espacio de nombres. Este es el nivel mayor de cobertura o alcance, aunque en este nivel solo podemos declarar tipos como clases, estructuras y enumeraciones, ya que los procedimientos solamente se pueden declarar dentro de un tipo.

Accesibilidad

Ya sean clases o miembros de las clases, podemos darle diferentes tipos de accesibilidad. Los modificadores de accesibilidad son:

Modificador (VB)	**Descripción**
Dim	Declara una variable en un módulo, procedimiento o bloque. Cuando se usa para declarar una variable a nivel de módulo, se puede sustituir por Private
Private	El elemento declarado sólo es visible dentro del nivel en el que se ha declarado.
Public	El elemento es visible en cualquier parte.
Friend	El elemento es visible dentro del propio ensamblado (proyecto).
Protected	El elemento es visible sólo en las clases derivadas.
Protected Friend	El elemento es visible en las clases derivadas y en el mismo ensamblado.

Las variables declaradas dentro de un procedimiento solo son accesibles dentro de ese procedimiento, en este caso solo se puede aplicar el ámbito privado, aunque no podremos usar la instrucción Private, sino Dim o Static.

La accesibilidad predeterminada de cada tipo (clase, estructura, etc.), así como de las variables declaradas con Dim y de los procedimientos en los que no se indican el modificador de accesibilidad.

Podemos declarar tipos dentro de otros tipos, por tanto el ámbito y accesibilidad de esos tipos dependen del ámbito y accesibilidad del tipo que los contiene.